NASA/CR-2008-215549

Wake Turbulence Mitigation for Departures (WTMD) Prototype System

Software Design Document

James L. Sturdy
Raytheon Company, Hampton, Virginia

December 2008

NASA/CR-2008-215549

Wake Turbulence Mitigation for Departures (WTMD) Prototype System

Software Design Document

James L. Sturdy
Raytheon Company, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Task Order L70750D

December 2008

Available from:

NASA Center for AeroSpace Information (CASI)
7115 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 605-6000

SOFTWARE DESIGN DOCUMENT

For

Wake Turbulence Mitigation for Departures (WTMD) Prototype System

Revision History

Revision/Date	**Purpose of Revision**
25 January 2008	Preliminary Review Version
7 April 2008	Full Initial Review Version
17 September 2008	Final Version for External Release

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1 Introduction 6
  1.1 Scope 6
  1.2 Background 6
  1.3 Document Overview 7
  1.4 Referenced Documents 7
  1.5 Notes 8
    1.5.1 Acronyms 8
2 Requirements 9
3 Design Goals 10
4 Prototype Context 11
  4.1 WTMD Prototype Inputs 12
    4.1.1 ASOS Input 12
    4.1.2 Wind Profile Input 12
    4.1.3 Client Command and Status Input 12
    4.1.4 Playback Control Input 13
  4.2 WTMD Prototype Outputs 13
    4.2.1 WTMD Output 13
    4.2.2 WFA Output 14
5 Logical Architecture 15
  5.1 Processes 15
    5.1.1 Core Process 15
    5.1.2 Supervisor Display 15
    5.1.3 Local Controller Display 16
    5.1.4 WTMD-IDS Gateway 16
  5.2 Software Architecture 16
    5.2.1 Overall Structure 16
    5.2.2 Distributed Processing Support 17
    5.2.3 Communications Encapsulation 17
6 Classes 18
  6.1 Core Process Classes 18

6.1.1 WTMD_CoreApp 18
6.1.2 WfaWrapper 24
6.1.3 CoreAppWindow 28
6.1.4 UpdateTimer Class Reference 29
6.1.5 CoreDialog 30
6.2 Supervisor Display Process Classes 33
6.2.1 WTMD_SupervisorApp 33
6.2.2 SupervisorAppWindow 36
6.2.3 SupervisorDialog 37
6.2.4 SupervisorDialog::UpdateTimer 43
6.3 Local Display Process Classes 44
6.3.1 WTMD_LocalApp 44
6.3.2 LocalAppWindow 47
6.3.3 LocalDialog 48
6.3.4 LocalDialog::UpdateTimer 52
6.4 WTMD-IDS Gateway Process Classes 53
6.4.1 WTMD_IDSGwApp 53
6.4.2 GwUpdateTimer Class Reference 57
6.5 Component Classes 58
6.5.1 Runway Status 58
6.5.2 System State Manager 71
6.5.3 Communication Manager 88
6.5.4 Messages 98
6.5.5 Clocks 120
6.5.6 Socket Wrappers 126
6.5.7 Physical Quantity Encapsulation 153
6.6 Index to Classes, Methods, and Functions 179

List of Figures

Figure 1: WTMD Prototype Context 11
Figure 2: WTMD Processes 15

1 Introduction

1.1 Scope

This document describes the software design of a prototype Wake Turbulence Mitigation for Departures (WTMD) system that was evaluated in shadow mode operation at the Saint Louis (KSTL) and Houston (KIAH) airports. This document describes the software that provides the system framework, communications, user displays, and hosts the Wind Forecasting Algorithm (WFA) software developed by the M.I.T. Lincoln Laboratory (MIT-LL). The WFA algorithms and software are described in a separate document produced by MIT-LL.

1.2 Background

The concept for the WTMD system is predicated on avoidance of wake turbulence and does not rely on the existence of an acceptable level of wake encounter. In its simplest form, the WTMD system consists of an automation tool that monitors a set of meteorological inputs to predict, with a high degree of certainty, that wakes generated by aircraft departing from a parallel runway can not impact aircraft departing from an upwind runway[1]. When such conditions exist, we say that the runway of interest is a wake independent runway (WIR). When this occurs, the Tower Supervisor may elect to enable the WTMD procedure. The WTMD system provides an indication of WTMD procedure status (ON (enabled) or OFF) of each runway to the air traffic controller(s) which are issuing takeoff clearances in the tower cab (Local Controller). Before the local controller decides to apply the wake separation delay to a departure after the departure of a B757 or heavy from the parallel runway, he would consider the WTMD procedure status. If the WTMD procedure is ON for the runway of interest, the wake turbulence separation standards following a B757 or heavy departing the parallel runway are not required for the WTMD ON runway. Otherwise, the current wake separation standards are implemented. This assumes, of course, that neither the preceding departure nor the departure of interest will be turning towards the parallel runway after liftoff. This concept never requires the controller to release a CSPR departure without delay; it authorizes the controller to apply departure standards that do not include a delay for wake separation for aircraft departing on the parallel runway when a positive WIR status exists for the departure runway

The Tower Supervisor will enable the WTMD system based on an expectation that favorable weather conditions will exist for an operationally significant period based on analysis of existing meteorological products and/or consultation with weather services unit personnel. The Supervisor will inform the local controllers whenever WTMD operations will be enabled or terminated. The WTMD system will produce a WIR status indication for each departure runway that is continuously updated at least once a minute based on the measured surface wind and rapid update cycle (RUC) forecasted winds

[1] This includes not just the present moment but also the time required for either the second departure to achieve current separation standards from the preceding B757/Heavy or the 2-3 minute currently required wake delay, whichever is shorter.

aloft. Because it enables a procedure that replaces the current 2-3 minute wake mitigation departure hold, the positive WIR status indication will be considered safety critical and meet appropriate performance requirements for integrity[2]. Anytime weather conditions not meeting preset criteria for WTMD operations are detected or predicted to occur within the system look-ahead period (approximately 20 minutes) or when there is an internal fault, the WTMD system will remove the positive WIR for affected departure runways. If this happens when the procedure had been enabled, the system removes the WTMD ON indication and generates an appropriate alert as well. Aircraft departing with a reduced delay at the time the WTMD ON status is removed will be permitted to continue since they can safely attain current wake separation standards based on the previous positive WIR status. Following any system initiated removal of the WTMD ON status, only the tower supervisor may re-enable the WTMD ON indication, assuming a positive WIR status exists. The tower supervisor may disable a WTMD ON indication or the entire WTMD system whenever desired.

1.3 Document Overview

This Software Design Document defines the software design of the WTMD prototype used at the Saint Lois (KSTL) and Houston (KIAH) airports. This document is limited to the WTMD prototype software. The reader should refer to the referenced documents for a full understanding of the WTMD system and its concept of operations.

Section 2 briefly discusses the WTMD requirements. Section 3 provides a brief description of the primary design goals for the WTMD prototype architecture. Section 4 presents the context within which the WTMD prototype operates, focusing on inputs and outputs. Section 5 describes the processes and overall software architecture used to implement the WTMD prototype. Section 6 provides the detailed description of each major class used in the design.

1.4 Referenced Documents

The following documents form a part of this specification to the extent referenced herein.

Doc. No.	Title	Date	Source
	Preliminary Program Requirements for Closely Spaced Parallel Runway Departure Capacity Improvement (CSPR DCI) Version 1.0	21 May 2007	FAA
	Program Requirements for Wake Turbulence Mitigation for Departures (WTMD) System Version 1.4	27 June 2008	FAA

[2] This determination applies only to those WTMD system components responsible for the positive WIR indication. A FMEA will be performed to appropriate design assurance categories for each system component.

Doc. No.	Title	Date	Source
MP 05W0000285 R2	Initial Concept of Use for Closely Spaced Parallel Runways (CSPR) Wind-Dependent Departures	August 2008	MITRE
F045-B06-0511	Initial Functional Architecture for Wake Turbulence Mitigation for Departures (WTMD): Functional Flow Diagrams	May 2006	MITRE
MP 06W0000097	Initial Functional Architecture for Wake Turbulence Mitigation for Departures (WTMD): Functional Descriptions	May 2006	MITRE
43PM Wx-0105	Wind Forecast Algorithm for Wake Mitigation for Departures (WTMD): Baseline algorithm description and performance summary	10 July 2008	MIT-LL

1.5 Notes

1.5.1 Acronyms

ASOS	Automated Surface Observing System
FAA	Federal Aviation Administration
FMEA	Failure Modes and Effects Analysis
GUI	Graphical User Interface
ITWS	Integrated Terminal Weather System
MIT-LL	Massachusetts Institute of Technology Lincoln Laboratory
RUC	Rapid Update Cycle (a gridded near-term weather forecast product from the National Weather Service)
SDD	Software Design Document
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WFA	Wind Forecasting Algorithm
WIR	Wake Independent Runway
WTMD	Wake Turbulence Mitigation for Departures

2 Requirements

The WTMD prototype software is intended to host the WFA for real-time demonstration at field sites and at NASA facilities. The key requirements that the prototype was intended to meet are summarized below. The referenced requirement documents provide a more complete set, though the prototype was only expected to meet the basic functionality implied by the complete requirement set.

The prototype shall accept pre-processed ASOS and RUC wind data from live feeds via a TCP connection over the internet to data servers operated by MIT-LL.

The prototype shall support running from recorded data inputs in a playback mode.

The prototype shall include a representative Tower Supervisor display. The Tower Supervisor display shall output system, WIR and WTMD status.

The Tower Supervisor Display shall accept commands to enable or disable the WTMD ON status for a runway.

The prototype shall include a representative Local Controller display.

The Local Controller display shall output the system and WTMD status.

The Local Controller display shall accept commands to silence a WTMD alert.

The prototype shall also output WTMD status via a serial interface for display on an IDS4 or ACE-IDS5.

The prototype shall provide an audible alert whenever WTMD has been enabled for a runway but the system or WIR status does not support a WTMD ON status

3 Design Goals

The architecture of the WTMD prototype software is intended to serve the following purposes:

1. Provide a clean and maintainable software architecture that demonstrates segregation of critical and non-critical functions and is easily extended to support additional display concepts.
2. Minimize the need to modify the WFA source code to maximize commonality with the WFA software baseline being developed by MIT-LL.
3. Minimize the impact of adding, modifying, or replacing displays on the core implementation of the WFA.
4. Provide object location transparency to facilitate changing the allocation of components to processes and processors.

4 Prototype Context

As shown in Figure 1, the WTMD prototype core receives two data feeds via a pair of TCP connections to a server at MIT-LL. One feed carries ASOS sensor data. The second carries relevant wind profile data grid points extracted from the latest RUC dataset by an application running at MIT-LL[3]. This RUC processing is similar in nature to the centralized processing of RUC data that the FAA performs to feed a subset of the data to individual ITWS field sites. The data in each feed is encoded and decoded by a weather-object library developed by MIT-LL. The prototype provides representations of displays suitable for Tower Supervisors, Local Controllers, and maintenance personnel. It also includes provisions for sending data to external display systems such as IDS4 and ACE-IDS5.

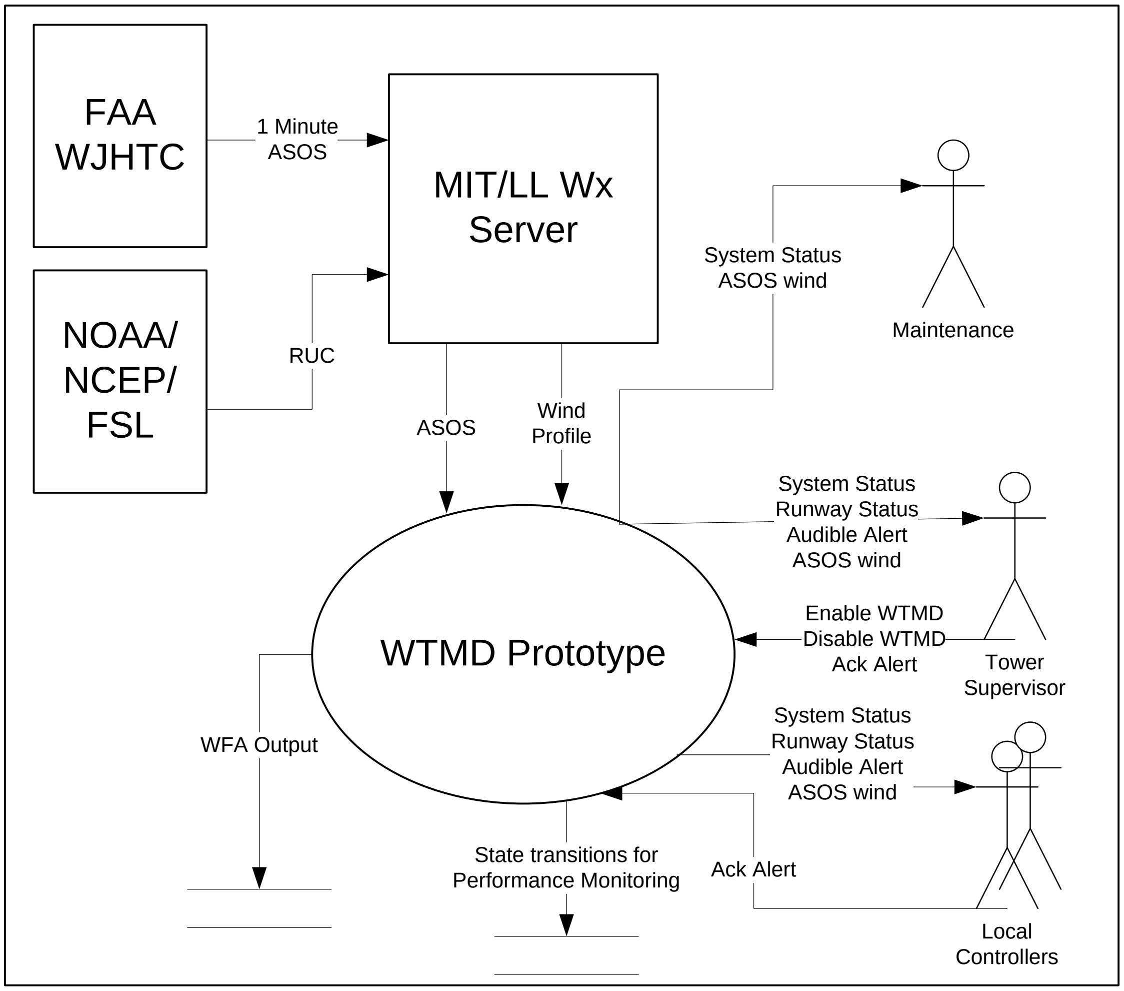


Figure 1: WTMD Prototype Context

[3] Though this RUC processing is external to the WTMD prototype's context, it would likely be considered a component of a deployed WTMD system and thus internal to the deployed system's context.

The WTMD prototype consists of a core process and one or more clients. The core process handles Transmission Control Protocol (TCP) communications from the MIT-LL weather data server (or playback data server) and maintains the system, WIR, and WTMD statuses. The clients implement user interfaces or format and send data to external systems.

Internal to the prototype context, the WTMD prototype core listens on a User Datagram Protocol (UDP) socket to receive command and status messages from the client processes. When the core detects a new client through the receipt of a client status message from a previously unknown client, the core opens a UDP output socket for sending data to that client, using host and port information contained in the status message.

4.1 WTMD Prototype Inputs

The WTMD prototype receives input from ASOS and RUC weather data servers and accepts user inputs to enable or disable the WTMD procedure and to acknowledge alerts. There is an additional playback control input that is only active when the prototype is configured to run in playback (non real-time) mode.

4.1.1 ASOS Input

From the ASOS feed, the WTMD prototype uses the configured station identifier to select messages from the relevant site. From these relevant messages, the system extracts data fields for observation time, measured horizontal wind speed, and measured wind direction.

4.1.2 Wind Profile Input

From the wind profile feed, the WTMD prototype uses the configured airport identifier to select relevant messages[4]. From these relevant wind profile messages, the prototype uses data fields for forecast time of validity, forecast delta time (time of applicability is time of validity plus delta), grid point latitude, grid point longitude, forecast heights, and wind U and V components for each reported height.

4.1.3 Client Command and Status Input

The WTMD prototype core accepts messages from display clients via a UDP port. The prototype core host machine name and port number are specified to each client at client startup. The core accepts three message types from the client: client status, enable WTMD, and disable WTMD.

The client status message contains the client name, an enumeration for the client status, the client's processor host-name, the client's configured input port number, and a text

[4] Each message contains data from a small set of RUC grid points associated in configuration data with a specific airport. The RUC pre-processor at MIT-LL extracts these relevant grid points for multiple WTMD airports.

description of any client fault. The core expects to receive a status message from each client every 10 seconds (a fault is declared after 12 seconds and the client is discarded after 30 seconds of silence – the prototype clients send the message no less than once every 5 seconds).

The enable WTMD message includes the runway identifier that the procedure is to be enabled on.

The disable WTMD message also includes the runway identifier that the procedure is to be disabled on. The disable WTMD message also serves as the acknowledgement for an active alert status on that runway.

4.1.4 Playback Control Input

The playback control input, which is only present when the WTMD prototype is started in a special playback mode, consists of a secondary control message that is inserted into the ASOS and RUC data streams by the playback data server. This message contains:

1. a pair of times representing a wall-clock time that the message is effective and a playback data timestamp that corresponds to the wall-clock time,
2. a playback rate (how fast does the playback data timestamp progress relative to the wall clock),
3. a flag that signifies that the data timestamp has been jumped (signaling a discontinuity in the data feed that would necessitate resetting the WFA data structures), and
4. a flag that indicates the system should terminate.

These inputs permit the WTMD prototype to remain synchronized with the timestamps in the recorded data.

4.2 WTMD Prototype Outputs

The WTMD prototype has two primary outputs: the WTMD data used by displays and the WFA output data stream that contains all of the engineering data relevant for monitoring the internal state of the WFA algorithms. In the WTMD prototype, the WFA output is only used to archive data for later comparison with the equivalent data produced by a prototype operating at MIT-LL. Timestamped system and runway state transitions are also recorded to disk to support off-line analysis of system performance.

4.2.1 WTMD Output

The WTMD prototype sends output data to each client using a UDP socket interface. The client host and port numbers used for this output are specified by each client in the client status messages sent periodically by each client. This output carries five messages: core status, runway status, fault log element, fault log, and ASOS winds.

4.2.1.1 Core Status WTMD Output

The core status message contains an enumeration for the system state (INITIALIZING, OPERATIONAL, FAILED, and SHUTDOWN) and a list of text descriptions for any active faults. This message, which also serves as a core heartbeat to the clients, is sent whenever the information changes or every half-second if the information is static (clients declare a fault when the core status data has not been received within the past 12 seconds).

4.2.1.2 Runway Status WTMD Output

The runway status message contains a set of runway statuses, one for each configured runway. The runway status contains a textual runway identifier, a flag indicating whether or not the runway is available and a flag indicating whether the WTMD procedure has been enabled for that runway. This message is sent whenever the information changes and is resent every 5 seconds if the information remains unchanged.

4.2.1.3 Fault Log Element WTMD Output

The fault log element message contains a record of a previously active fault, which is represented by a fault description, the time that the fault was first observed, and the time that the fault was cleared. This message is sent whenever a fault is cleared in the prototype core in order to incrementally update the fault logs held by each client.

4.2.1.4 Fault Log WTMD Output

The fault log message contains a complete set of previously active faults up to a configured maximum (currently, only the 20 most recent are retained). Each entry in the set is a fault log element as described above. The core sends this message whenever the core detects a new client to synchronize the log held by the new client.

4.2.1.5 ASOS Winds WTMD Output

This message contains the most recent wind speed, direction, and time of measurement received from the configured ASOS station. The core sends this message whenever the core receives a new relevant ASOS message from the weather data server. This message allows clients that do not already have a source of wind information to display the current surface winds.

4.2.2 WFA Output

The WTMD prototype also produces an output stream containing detailed data generated by the WFA algorithm which is primarily intended for engineering assessment of the WFA operation. This data is available as a MIT-LL defined weather message object via a MIT-LL defined weather stream interface. The message is sent after processing an ASOS or RUC update.

5 Logical Architecture

5.1 Processes

The WTMD prototype currently includes three different display processes and the core process that was the focus of Section 4. These processes are depicted in Figure 2. The display processes, each of which is an instance of a WTMD client, are the supervisor display, local controller display, and WTMD-IDS gateway.

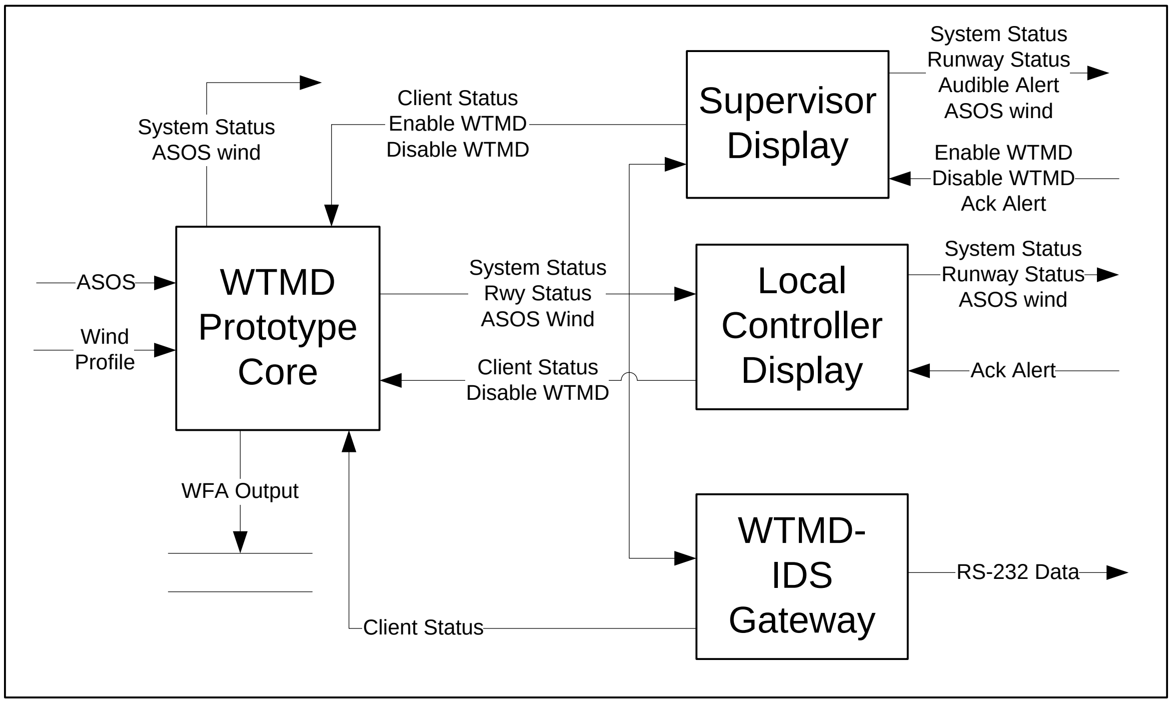


Figure 2: WTMD Processes

5.1.1 Core Process

The core process serves as the central state repository for the WTMD prototype and manages the connections to the external sources of ASOS and RUC data (as well as the playback control when configured for playback operation). The core does provide a user display for more maintenance-oriented information such as current time, system status, active faults, fault log, and last ASOS received, but its primary function is to manage the data feeds, to manage the WFA code, and to maintain WTMD system state.

5.1.2 Supervisor Display

The supervisor display is a representation of the data and controls that a tower supervisor would need to operate the system. It shows the status of the overall WTMD system and of each runway. When the system is operational and a runway is available and no runways are enabled, it provides a user control for enabling the WTMD procedure. When the procedure has been enabled, it provides an input control for disabling the procedure (if the runway is still available) or acknowledging the alert (if the runway is no longer available). If the system status indicates a fault, the supervisor display provides a control for looking at the fault information. The supervisor display provides a control for looking at the log of the most recent cleared faults and for looking at the list of active faults when faults are active. In the prototype, the supervisor display generates the audible alert, though this function could have been hosted in a separate process or multiple processes.

5.1.3 Local Controller Display

The local controller display is a representation of the data and controls that might be available to a local controller. As such, it is simpler and more limited than the supervisor display, though the underlying software is mostly in common. The local controller display only shows the WTMD system status and a message that the WTMD procedure is either off, enabled for a particular runway, or in an alert state for a particular runway. If the procedure is in an alert state, the local controller display will provide an input control for acknowledging the alert.

5.1.4 WTMD-IDS Gateway

The WTMD-IDS gateway is not, strictly speaking, a display itself, but serves as a translator between the WTMD prototype and an IDS system connected to the prototype via an RS-232 serial port. The gateway takes the system and runway status information and converts them into a text message that it formats according to the IDS system interface specification and writes to the appropriate serial port. At this time, the IDS interface does not provide any mechanism for injecting control inputs back into the WTMD prototype

5.2 Software Architecture

5.2.1 Overall Structure

The WTMD software architecture uses the Model-View-Controller (MVC) design pattern to separate the model (system state and runway status) from the details of the user interface(s) that are visible to the observers. The model consists of two components built on the singleton pattern: `RwyStatusMgr,` which maintains a list of `RunwayStatus` objects and `SystemStateMgr,` which maintains the system state (an enumeration with values `FAILED`, `INITIALIZING`, `OPERATIONAL`, and `SHUTDOWN`) and a list of `Faults` which consist of a string and a start and end time for use in generating a fault log. The `RunwayStatus` object stores the runway identifier as well as Boolean parameters to indicate whether or not the runway is available (`available_`) and whether or not the procedure has been enabled (`enabled_)`. The WTMD procedure is ON for a runway if the status indicates the runway is both available and enabled. A runway is in the ALERT state if the runway is enabled but not available. The alert is acknowledged by setting `enabled_` to false. A `RunwayStatus` will not permit its `available_` parameter to be set true if the `enabled_` parameter is already true.

5.2.2 Distributed Processing Support

Because the WTMD prototype needs to provide flexibility in the number and nature of displays supported, the logical architecture has support for distributed processing built into it. The proxy design pattern is used to achieve this by providing a means for the information content of the model components and the control inputs to the model components to be replicated between processes without requiring the display implementations to manage the complexity of communication protocols. This is accomplished by deriving subclasses for each of the model components (`SystemStateMgr` and `RwyStatusMgr`) to encapsulate the message passing appropriate for use in the core process (`CoreSystemStateMgr` and `CoreRwyStatusMgr`) or a client process (`ClientSystemStateMgr` and `ClientRwyStatusMgr`). These classes take care of monitoring the input channel for appropriate commands (core side) and state updates (client side). They also re-implement methods that change the model state so that either the modified data is replicated to clients (core side) or transmitted back to the core (client side).

5.2.3 Communications Encapsulation

A communications manager (`ComMgr`) encapsulates the management of the UDP sockets used for the underlying communications. This class provides methods for collecting input messages from the communication link, transmitting data to the communication link, and returning a list of messages received. Specialized versions of this class serve the core and client processes. The `CoreComMgr` initializes the input UDP port for accepting client messages and also maintains the list of client connections and implements the send method so that a copy of all output messages is sent to all established clients. The `ClientComMgr` initializes the input UDP port for accepting core messages and establishes the output connection to the core's UDP input port. These classes also ensure that the communication links are cleaned up on termination.

A `Msg` class is the base class for all of the different message types communicated between processes. This class provides a common interface for converting message objects to a byte stream for transmission and for reconstructing message objects of the correct type from a received byte stream. There is a derived message class corresponding to each of the message types identified in Sections 4.1.3 and 4.2.1.

6 Classes

This section provides detailed descriptions of the methods and attributes of each of the major classes in the prototype. Sections 6.1 through 6.4 list the classes that are specific to a particular process. Section 6.5 contains descriptions for the classes that are less process specific and are potentially shared by multiple processes. Section 6.6 provides an index of class and method names.

Not documented within these subsections are classes from the JUCE display object library, the ACES system call encapsulation library, and the MIT-LL WFA objects and Wx weather object library and their associated MIT-LL generated support libraries.

6.1 Core Process Classes

The main application object for the WTMD core process is `WTMD_CoreApp`. It is responsible for creating all of the other primary application components and implementing the main periodic processing loop which is invoked through the `UpdateTimer` class, which is a specialization of the JUCE `Timer` class. The WFA algorithm code itself is encapsulated by the `WfaWrapper` class. `CoreAPPWindow` is the JUCE window class for the core process GUI. The `CoreDialog` class implements the look and behavior of the GUI within the window frame.

6.1.1 WTMD_CoreApp

Public Member Functions

- **WTMD_CoreApp** ()
- virtual **~WTMD_CoreApp** ()
- void **update** ()
- void **handle** (WxSaObsOneMinASOS &obj)
- void **handle** (WxWVProfiles &obj)
- void **handle** (WxTimeCtl &obj)

  JUCE Methods

  - virtual void **initialise** (const String &command_line_parms)
  - virtual void **shutdown** ()
  - virtual const String **getApplicationName** ()
  - virtual const String **getApplicationVersion** ()
  - virtual bool **moreThanOneInstanceAllowed** ()
  - virtual void **systemRequestedQuit** ()
  - virtual void **unhandledException** (const std::exception *e, const String &sourceFile, const int lineNumber)

Static Public Member Functions

- static **WTMD_CoreApp** & **instance** ()

Private Member Functions

- **WTMD_CoreApp** (const **WTMD_CoreApp** &)

- **WTMD_CoreApp** & **operator=** (const **WTMD_CoreApp** &)

Private Operation Report Pieces

- void **closeoutReport** ()
- std::ofstream * **report_stream_**
- bool **was_available_**
- **WakeVAS::AbsoluteTime available_time_**
- **WakeVAS::RelativeTime available_duration_total_**
- bool **was_operational_**
- **WakeVAS::AbsoluteTime operational_time_**
- **WakeVAS::RelativeTime operational_duration_total_**

Private Attributes

I/O Pieces

- bool **offline_**
- WxArchiveIStream * **asos_istream_**
- WxArchiveIStream * **profile_istream_**
- WxArchiveOStream * **alert_ostream_**

Primary Application Components

- **WakeVAS::CoreComMgr** * **com_mgr_**
- **WakeVAS::SimClock** * **clock_**
- **WakeVAS::CoreSystemStateMgr** * **state_mgr_**
- **WakeVAS::CoreRwyStatusMgr** * **rwy_mgr_**
- **WakeVAS::WfaWrapper** * **wfa_**
- **CoreAppWindow** * **window_**
- **CoreDialog** * **view_**
- **UpdateTimer** * **updater_**

Static Private Attributes

- static **WTMD_CoreApp** * **instance_** = NULL

Implementation Variables

- static const unsigned int **TIMER_INTERVAL** = 50

Detailed Description

This is the main application object for the WTMD core process. It is responsible for creating all of the primary application class instances (components) and performing the top-level operations. This class re-implements the singleton pattern so other components can access the non-JUCE methods.

Constructor & Destructor Documentation

WTMD_CoreApp ()

Constructor sets the instance pointer and will assert if another instance already exists. Performs only default initialization of members -- full initialization is performed by **`initialise()`**.

~WTMD_CoreApp () `[virtual]`

Destructor clears the instance pointer. Object cleanup is performed in **`shutdown()`**.

WTMD_CoreApp (const WTMD_CoreApp &) `[private]`

Not allowed and not implemented.

Member Function Documentation

WTMD_CoreApp & instance () `[static]`

Return the singleton instance. Asserts if instance has not been created.

void update ()

This is the head of the periodic processing main loop. It starts by having the `ComMgr` gather input from clients. Then it checks for input from the WxStreams (which will cause the calling of the appropriate message object handlers), or if offline, generates a dummy ASOS message. Next, it updates the `WfaWrapper`, `RwyStatusMgr`, `SystemStateMgr`, and view and finishes by telling the `ComMgr` to flush output. Finally, if summary report generation has been enabled, it checks for any state transitions and writes to the report file if appropriate.

void handle (WxSaObsOneMinASOS & *obj*)

The weather object is passed to the `WfaWrapper` for processing. If it was used (i.e. came from a relevant station) and the data is ok, the wind data is also sent to clients and used to update the local view.

void handle (WxWVProfiles & *obj*)

The weather object is passed to the `WfaWrapper` for processing.

void handle (WxTimeCtl & *obj*)

The time control object (which only comes during playback) is examined. If it differs from the last one processed (the same data can come from multiple streams), and it indicates a time jump, the stateful components are reset in preparation for picking up the data stream at a new point. The clock information in the message is used to set the local time source.

void initialise (const String & *command_line_parms*) `[virtual]`

This method parses the command line to extract options, checks for usage violations, initializes stream connections, and then constructs the primary application objects. Then, it creates the application window and the timer that triggers periodic updates.

void shutdown () `[virtual]`

If the `state_mgr_` has been created, **`shutdown()`** begins by signalling clients to shut down, then it closes the summary report and deletes the primary application objects in the reverse order of their creation.

const String getApplicationName () `[virtual]`

Return the name of the application.

const String getApplicationVersion () `[virtual]`

Return the version number of the application.

bool moreThanOneInstanceAllowed () `[virtual]`

Return `true`.

void systemRequestedQuit () `[virtual]`

Handle a request to quit from the window manager. A confirmation dialog is displayed and, if confirmed, `quit()` is called to initiate the shutdown sequence.

void unhandledException (const std::exception * *e*, const String & *sourceFile*, const int *lineNumber*) `[virtual]`

Default JUCE handler for an unhandled exception. This implementation merely attempts to print the file and line number information.

WTMD_CoreApp& operator= (const WTMD_CoreApp &) `[private]`

Not allowed and not implemented.

void closeoutReport () `[private]`

Print final statistics and reset report pieces for a new interval.

Member Data Documentation

bool offline_ `[private]`

`true` iff not getting data from WxStreams.

WxArchiveIStream* asos_istream_ `[private]`

WxStream for receiving ASOS messages. Only `NULL` if `offline_` is true.

WxArchiveIStream* profile_istream_ `[private]`

WxStream for receiving wind profile (RUC) messages. Only `NULL` if `offline_` is true.

WxArchiveOStream* alert_ostream_ `[private]`

WxStream to which the final WFA output is sent (may be `NULL`).

std::ofstream* report_stream_ `[private]`

Stream to which summary report info is written. `NULL` if reporting was not enabled on command line.

bool was_available_ `[private]`

At least one runway was available on last pass.

WakeVAS::AbsoluteTime available_time_ `[private]`

When a runway first became available.

WakeVAS::RelativeTime available_duration_total_ `[private]`

Running total of all time a runway was available.

bool was_operational_ `[private]`

The WTMD system was available on last pass.

WakeVAS::AbsoluteTime operational_time_ `[private]`

When the system first became operational (since last fault).

WakeVAS::RelativeTime operational_duration_total_ `[private]`

Running total of all time the system was operational.

WakeVAS::CoreComMgr* com_mgr_ `[private]`

The communications manager.

WakeVAS::SimClock* clock_ `[private]`

The local time source. A `SimClock` is used to support playback.

WakeVAS::CoreSystemStateMgr* state_mgr_ `[private]`

The system state manager.

WakeVAS::CoreRwyStatusMgr* rwy_mgr_ `[private]`

The runway status manager.

WakeVAS::WfaWrapper* wfa_ `[private]`

The WFA wrapper instance.

CoreAppWindow* window_ `[private]`

The JUCE application window.

UpdateTimer* updater_ `[private]`

The JUCE view managed by the window. Be carefull to `NULL` when `window_` is destroyed! The JUCE update timer.

WTMD_CoreApp * instance_ = NULL `[static, private]`

The singleton instance.

Implementation Variable Documentation

const unsigned int TIMER_INTERVAL = 50 `[static]`

Time interval for triggering periodic updates in msec.

6.1.2 WfaWrapper

Public Member Functions

- **WfaWrapper** ()
- **~WfaWrapper** ()
- bool **initialize** (const char *pred_config_file_name, const char *alert_config_file_name)
- void **reset** ()
- void **update** (const **WakeVAS::AbsoluteTime** &now, WxArchiveOStream *alert_ostream)
- bool **update** (WxSaObsOneMinASOS &inAsos)
- void **update** (WxWVProfiles &inProf)
- void **update** (const **WakeVAS::Angle** &asos_dir_from, const **WakeVAS::Speed** &asos_speed, const **WakeVAS::AbsoluteTime** &time)

Static Public Member Functions

- static **WfaWrapper** & **getInstance** ()

Private Member Functions

- **WfaWrapper** (const **WfaWrapper** &rhs)
- **WfaWrapper** & **operator=** (const **WfaWrapper** &rhs)
- void **update** (WxWVWinds &inWind)

Private Attributes

- **WakeVAS::AbsoluteTime init_time_**
- bool **ruc_initializing_**

Static Private Attributes

- static **WfaWrapper** * **instance_** = NULL

Implementation Variables

- static const int **MAX_ASOS_AGE_SECS** = 240
- static const **RelativeTime MAX_ASOS_AGE** (**MAX_ASOS_AGE_SECS**, RelativeTime::SECONDS)
- static const **RelativeTime MAX_RUC_AGE** (1.2, RelativeTime::HOURS)
- static vector< WindPredParams > **windPredParams**
- static vector< WindPredData > **windPredData**
- static map< string, WVAlertParams > **alertParams**
- static map< string, AlertData > **alertDataSet**
- static WxWVProfiles **currProf**
- static WxWVWinds **currWind**
- static std::string **siteID**

Detailed Description

This singleton object encapsulates the MIT Lincoln Laboratory Wind Forecast Algorithm (WFA) code. It coordinates passing new data into the WFA and extracts runway availability and system state (initializing, stale data, etc.) information from the WFA.

Constructor & Destructor Documentation

WfaWrapper ()

Constructor sets the instance pointer and will assert if another instance already exists. Performs only default initialization of members -- full initialization is performed by **`initialize()`** and **`reset()`**.

~WfaWrapper ()

Destructor clears the instance pointer.

WfaWrapper (const WfaWrapper & *rhs*) `[private]`

Not allowed and not implemented.

Member Function Documentation

WfaWrapper & getInstance () `[static]`

Return the singleton instance. Asserts if instance has not been created.

bool initialize (const char * *pred_config_file_name*, const char * *alert_config_file_name*)

Uses the named files to initialize the WFA parameter objects. Use this set of configured runways to initialize runway structures in **`CoreRwyStatusMgr`**. Return true iff initialization succeeds.

void reset ()

Clear all WFA state data without changing parameter configuration. Also sets the `init_time_` and `ruc_initializing_` members.

void update (const WakeVAS::AbsoluteTime & *now*, WxArchiveOStream * *alert_ostream*)

This is the periodic update method which invokes the periodic `AlertData` WFA processing, gathers runway status data from the WFA, and checks for stale or missing data. It will update the rwy available status held by **`RwyStatusMgr`** and may change the system state and faults held by **`SystemStateMgr`**.

bool update (WxSaObsOneMinASOS & *inAsos*)

If the ASOS report is from the configured site, it is passed to the WFA surface wind prediction algorithm for processing. The results are then passed via `update (WxWVWinds&)` for WFA alert data processing. The actual runway status will not be changed until the periodic update method is called. Returns true if report was used.

void update (WxWVProfiles & *inProf*)

If the RUC profile is for the configured site, it is passed to the WFA alert data processing function. The actual runway status will not be changed until the periodic update method is called.

WfaWrapper& operator= (const WfaWrapper & *rhs*) `[private]`

Not allowed and not implemented.

void update (WxWVWinds & *inWind*) `[private]`

This update method replaces the stream interface between the wind prediction process and the alert process in the MIT-LL WFA architecture. This is a somewhat inefficient way of doing things when both functions are performed in the same process, but retaining the intermediary structure minimizes the need to change the MIT-LL WFA `AlertData` code.

Member Data Documentation

WakeVAS::AbsoluteTime init_time_ `[private]`

Time at which WFA wrapper was reset. Used to avoid premature declaration of stale ASOS or RUC data on startup.

bool ruc_initializing_ `[private]`

Flag used to monitor when still waiting for first RUC data after startup.

WfaWrapper * instance_ = NULL `[static, private]`

The singleton instance.

Implementation Variable Documentation

const int MAX_ASOS_AGE_SECS = 240 `[static]`

How long to wait before declaring ASOS data stale (based on time of measurment). This parameter needs to account for update interval (1 minute) plus an allowance for transmission latency. The Current value of 4 minutes is longer than desired, but the prototype receives

ASOS data through a long path with a number of network/server hops, so typical latency is 75 to 90 seconds with occasional longer delays.

const RelativeTime MAX_ASOS_AGE(MAX_ASOS_AGE_SECS, RelativeTime::SECONDS) `[static]`

`MAX_ASOS_AGE_SECS` expressed as a `RelativeTime`.

const RelativeTime MAX_RUC_AGE(1.2, RelativeTime::HOURS) `[static]`

How long to wait after startup before declaring RUC data stale. After startup, staleness is determined by lack of forecast data for required time periods.

vector<WindPredParams> windPredParams `[static]`

Set of WFA parameters for processing surface winds.

vector<WindPredData> windPredData `[static]`

Set of WFA data associated with processing surface winds, in same order as windPredParams.

map<string, WVAlertParams> alertParams `[static]`

Set of WFA parameters for alert (runway availability) determination indexed by runway name.

map<string, AlertData> alertDataSet `[static]`

Set of WFA data associated with alert (runway availability) determination indexed by runway name.

WxWVProfiles currProf `[static]`

Most recent RUC profile data processed.

WxWVWinds currWind `[static]`

Most recent surface wind prediction passed from surface wind processing to 'alert' processing.

std::string siteID `[static]`

Airport identifier used to select RUC messages of interest.

6.1.3 CoreAppWindow

Public Member Functions

- **CoreAppWindow** (**CoreDialog** *content, const String &title="Core")
- virtual **~CoreAppWindow** ()
- virtual void **closeButtonPressed** ()

Detailed Description

This class specializes the JUCE `DocumentWindow` to customize how it appears, which frame buttons are visible, capture pressing of the close button and to make the window appear on the taskbar.

Constructor & Destructor Documentation

CoreAppWindow (CoreDialog * *content*, const String & *title* = `"Core"`)

Constructor gets things properly initialized for the way we want the window to appear and behave.

~CoreAppWindow () `[virtual]`

Perform only implicit cleanup.

Member Function Documentation

void closeButtonPressed () `[virtual]`

Handle the pressing of the close button.

6.1.4 UpdateTimer Class Reference

Public Member Functions

- **UpdateTimer** (**WTMD_CoreApp** &parent)
- virtual void **timerCallback** ()

Private Attributes

- **WTMD_CoreApp** & **parent_**

Detailed Description

This class links the JUCE timer to the main application object in order to invoke the periodic update method.

Constructor & Destructor Documentation

UpdateTimer (WTMD_CoreApp & *parent*)

Constructor performs default initialization and stores a reference to the `WTMD_CoreAPP` for calling into later.

Member Function Documentation

virtual void timerCallback () `[virtual]`

This method reimplementation invokes the periodic update method on the main application object

---

Member Data Documentation

WTMD_CoreApp& parent_ `[private]`

The main application object reference.

6.1.5 CoreDialog

Public Member Functions

- **CoreDialog** (bool manual_asos)
- **WakeVAS::Angle getAsosDirection** () const
- **WakeVAS::Speed getAsosSpeed** () const
- void **setAsosDirection** (const **WakeVAS::Angle** &dir)
- void **setAsosSpeed** (const **WakeVAS::Speed** &speed)
- void **updateView** ()
- void **paint** (Graphics &g)
- void **resized** ()
- void **sliderValueChanged** (Slider *sliderThatWasMoved)
- void **buttonClicked** (Button *buttonThatWasClicked)

Private Member Functions

- void **handleQuitButton** ()
- void **handleViewLogButton** ()
- **CoreDialog** (const **CoreDialog** &)
- const **CoreDialog** & **operator=** (const **CoreDialog** &)

Private Attributes

- Label * **state_label_**
- Label * **state_detail_label_**
- Label * **system_state_**
- TextEditor * **state_detail_list_**
- Slider * **asos_direction_**
- Slider * **asos_speed_**
- Label * **asos_dir_label_**
- Label * **asos_speed_label_**
- TextButton * **shutdown_button_**
- Label * **time_label_**
- Label * **time_**
- TextButton * **view_log_button_**

Implementation Variables

- static const String **FAILED_SYSTEM_STATE_TEXT** ("Fault")
- static const String **INITIALIZING_SYSTEM_STATE_TEXT** ("Initializing")
- static const String **OPERATIONAL_SYSTEM_STATE_TEXT** ("Operational")

Detailed Description

This is the view and controller portion of the Model-View-Controller pattern used in the WTMD core process. The overall class framework is an auto- generated component, created by the Jucer GUI builder. The non-JUCE code appears within sections of the framework clearly delineated by comments.

Member Function Documentation

WakeVAS::Angle getAsosDirection () const

Return the value of the direction slider.

WakeVAS::Speed getAsosSpeed () const

Return the value of the speed slider.

void setAsosDirection (const WakeVAS::Angle & *dir*)

Set the value of the direction slider in tens of degrees. This control is only enabled if the `manual_asos` constructor argument was `true`.

void setAsosSpeed (const WakeVAS::Speed & *speed*)

Set the value of the speed slider in knots. This control is only enabled if the `manual_asos` constructor argument was `true`.

void updateView ()

Update the view elements according to the overall WTMD system state.

1. Determine the overall state text to display and set the text and color of the `system_state_` Label accordingly.
2. If the list of faults has changed, format a (potentially) multi-line text representation of the active faults and put the text in the `state_detail_list_`.
3. Format and display the current system clock time.

void handleQuitButton () `[private]`

Tell the application that the user has requested a shutdown.

void handleViewLogButton () `[private]`

Format a (potentially) multi-line text representation of the contents of the fault log maintained by the `SystemStateMgr` and create a JUCE modal dialog (`AlertWindow`) to display it.

Implementation Variable Documentation

static const String FAILED_SYSTEM_STATE_TEXT ("Fault") `[static]`

Text constant to display when system state is `FAILED`.

static const String INITIALIZING_SYSTEM_STATE_TEXT ("Initializing") `[static]`

Text constant to display when system state is `INITIALIZING`.

static const String OPERATIONAL_SYSTEM_STATE_TEXT ("Operational") `[static]`

Text constant to display when system state is `OPERATIONAL`.

6.2 Supervisor Display Process Classes

The main application object for the WTMD supervisor display process is `WTMD_SupervisorApp`. It is responsible for creating all of the other primary application components. `SupervisorAPPWindow` is the JUCE window class for the supervisor process GUI. The `SupervisorDialog` class implements the look and behavior of the GUI within the window frame and implements the main periodic processing loop which is invoked through the `SupervisorDialog::UpdateTimer` class, which is a specialization of the JUCE `Timer` class.

6.2.1 WTMD_SupervisorApp

Public Member Functions

- **WTMD_SupervisorApp** ()
- virtual **~WTMD_SupervisorApp** ()

  JUCE Methods

  - virtual void **initialise** (const String &command_line_parms)
  - virtual void **shutdown** ()
  - virtual const String **getApplicationName** ()
  - virtual const String **getApplicationVersion** ()
  - virtual bool **moreThanOneInstanceAllowed** ()
  - virtual void **systemRequestedQuit** ()
  - virtual void **unhandledException** (const std::exception *e, const String &sourceFile, const int lineNumber)

Private Member Functions

- **WTMD_SupervisorApp** (const **WTMD_SupervisorApp** &)
- **WTMD_SupervisorApp** & **operator=** (const **WTMD_SupervisorApp** &)

Private Attributes

- String **name_**

  Primary Application Components

  - **WakeVAS::ClientComMgr** * **com_mgr_**
  - **WakeVAS::SystemClock** * **clock_**
  - **WakeVAS::ClientSystemStateMgr** * **state_mgr_**
  - **WakeVAS::ClientRwyStatusMgr** * **rwy_mgr_**
  - **SupervisorAppWindow** * **window_**

Detailed Description

This is the main application object for the WTMD supervisor display process. It is responsible for creating all of the primary application class instances (components) and performing the top-level operations. This class inherits the singleton pattern.

Constructor & Destructor Documentation

WTMD_SupervisorApp ()

Constructor performs only default initialization of members -- full initialization is performed by **`initialise()`**.

~WTMD_SupervisorApp () `[virtual]`

Object cleanup is performed in **`shutdown()`**.

WTMD_SupervisorApp (const WTMD_SupervisorApp &) `[private]`

Not allowed and not implemented.

Member Function Documentation

void initialise (const String & *command_line_parms*) `[virtual]`

This method parses the command line to extract options, checks for usage violations, initializes stream connections, and then constructs the primary application objects. Then, it creates the application window. The periodic processing is performed by the **`SupervisorDialog`** which is created within the application window.

void shutdown () `[virtual]`

This method deletes the primary application objects in the reverse order of their creation.

const String getApplicationName () `[virtual]`

Return the name of the application.

const String getApplicationVersion () `[virtual]`

Return the version number of the application.

bool moreThanOneInstanceAllowed () `[virtual]`

Return `true`.

void systemRequestedQuit () `[virtual]`

Handle a request to quit from the window manager. A confirmation dialog is displayed with a message that indicates if any runways are enabled and, if confirmed, `quit()` is called to initiate the shutdown sequence.

void unhandledException (const std::exception * *e*, const String & *sourceFile*, const int *lineNumber*) `[virtual]`

Default JUCE handler for an unhandled exception. This implementation merely attempts to print the file and line number information.

WTMD_SupervisorApp& operator= (const WTMD_SupervisorApp &) `[private]`

Not allowed and not implemented.

Member Data Documentation

String name_ `[private]`

Process name defaults to "Supervisor", but can be overridden by command line options.

WakeVAS::ClientComMgr* com_mgr_ `[private]`

The communications manager.

WakeVAS::SystemClock* clock_ `[private]`

The local time source. Instantiated as a `RealtimeClock`.

WakeVAS::ClientSystemStateMgr* state_mgr_ `[private]`

The system state manager.

WakeVAS::ClientRwyStatusMgr* rwy_mgr_ `[private]`

The runway status manager.

SupervisorAppWindow* window_ `[private]`

The JUCE application window.

6.2.2 SupervisorAppWindow

Public Member Functions

- **SupervisorAppWindow** (const String &audio_file, const String &title="Supervisor")
- virtual **~SupervisorAppWindow** ()
- virtual void **closeButtonPressed** ()

Detailed Description

This class specializes the JUCE `DocumentWindow` to customize how it appears, which frame buttons are visible, capture pressing of the close button and to make the window appear on the taskbar.

Constructor & Destructor Documentation

SupervisorAppWindow (const String & *audio_file*, const String & *title* = `"Supervisor"`)

Constructor gets things properly initialized for the way we want the window to appear and behave.

~SupervisorAppWindow () `[virtual]`

Perform only implicit cleanup.

Member Function Documentation

void closeButtonPressed () `[virtual]`

Handle the pressing of the close button.

6.2.3 SupervisorDialog

Public Member Functions

- **SupervisorDialog** (const String &sound_file)
- void **paint** (Graphics &g)
- void **resized** ()
- void **buttonClicked** (Button *buttonThatWasClicked)

Private Classes

- class **UpdateTimer**

Private Types

- enum { **MAX_RWYS** = 4 }

Private Member Functions

- void **update** ()
- void **updateView** ()
- void **handleAsosButton** ()
- void **handleStateDetailButton** ()
- void **handleRwyButton** (const unsigned button_index)
- void **handleAudioButton** ()
- void **handleViewLogButton** ()
- void **enableAudioSystem** ()
- void **disableAudioSystem** ()
- **SupervisorDialog** (const **SupervisorDialog** &)
- const **SupervisorDialog** & **operator=** (const **SupervisorDialog** &)

Private Attributes

- **WakeVAS::Angle asos_direction_**
- **WakeVAS::Speed asos_speed_**
- String **audio_filename_**
- AudioSystem * **audio_system_**
- **UpdateTimer** * **updater_**
- Label * **state_label_**
- Label * **system_state_**
- TextButton * **asos_button_**
- TextButton * **state_detail_button_**
- TextButton * **rwy_button1_**
- TextButton * **rwy_button2_**
- TextButton * **rwy_button3_**
- TextButton * **rwy_button4_**
- Label * **rwy_label_**
- Label * **status_label_**
- Label * **rwy_id1_**
- Label * **rwy_id2_**
- Label * **rwy_id3_**

- Label * **rwy_id4_**
- Label * **rwy_status1_**
- Label * **rwy_status2_**
- Label * **rwy_status3_**
- Label * **rwy_status4_**
- Label * **asos_label_**
- Label * **asos_status_**
- TextButton * **audio_button_**
- Label * **audio_status_**
- TextButton * **view_log_button_**

Arrays of Per-runway Display Objects

- Label * **rwy_ids** [MAX_RWYS]
- TextButton * **rwy_buttons** [MAX_RWYS]
- Label * **rwy_statuses** [MAX_RWYS]

Implementation Variables

- static const unsigned int **TIMER_INTERVAL** = 100
- static const SocketInterface::SocketId **COORDINATION_PORT_NUM** = 8765
- static const String **ENABLE_BUTTON_TEXT** ("Enable")
- static const String **DISABLE_BUTTON_TEXT** ("Disable")
- static const String **SILENCE_BUTTON_TEXT** ("Acknowledge")
- static const String **BLANK_BUTTON_TEXT** ("")
- static const String **AVAILABLE_STATUS_TEXT** ("Available")
- static const String **UNAVAILABLE_STATUS_TEXT** ("OFF")
- static const String **AVAILABLE_AND_ENABLED_STATUS_TEXT** ("WTMD ON")
- static const String **ENABLED_ONLY_STATUS_TEXT** ("ALERT")
- static const String **BLANK_STATUS_TEXT** ("")
- static const String **FAILED_SYSTEM_STATE_TEXT** ("Fault")
- static const String **INITIALIZING_SYSTEM_STATE_TEXT** ("Initializing")
- static const String **OPERATIONAL_SYSTEM_STATE_TEXT** ("Operational")
- static const String **AUDIO_ENABLED_STATUS_TEXT** ("Audio alerts enabled")
- static const String **AUDIO_DISABLED_STATUS_TEXT** ("Audio alerts disabled")
- static const String **AUDIO_ENABLE_BUTTON_TEXT** ("Enable Audio")
- static const String **AUDIO_DISABLE_BUTTON_TEXT** ("Disable Audio")
- static const Colour **amber** (192, 128, 0)

Detailed Description

This is the view and controller portion of the Model-View-Controller pattern used in the WTMD supervisor process. The overall class framework is an auto- generated component, created by the Jucer GUI builder. The non-JUCE code appears within sections of the framework clearly delineated by comments.

Member Function Documentation

void update () `[private]`

This is the head of the periodic processing main loop. It performs the following steps:

1. Tell the `ComMgr` to gather input messages from the core process.
2. Update application components (`SystemStateMgr`, `RwyStatusMgr`). Check the system status to see if a shutdown has been commanded. Look for a new `AsosWindMsg`.
3. Update the view.
4. Tell the `ComMgr` to flush any output messages to the core process.

void updateView () `[private]`

Update the view elements according to the overall WTMD system state.

1. Determine the overall state text to display and set the text and color of the `system_state_` Label accordingly. If the state is not operational, no runway is considered available.
2. For each configured runway, set the Id, status, button text and button visibility. The button text is `ENABLE_BUTTON_TEXT` and the button is visible if the runway is available and no runways are enabled. The button text is `DISABLE_BUTTON_TEXT` and the button is visible if the runway is available and enabled. The button text is `ENABLED_ONLY_STATUS_TEXT` and the button is visible if the runway is enabled but not available (alert state). Otherwise, the button is hidden.
3. Update the wind speed and direction labels (which may be hidden if not enabled for display).

void handleAsosButton () `[private]`

Toggle the display of ASOS data.

void handleStateDetailButton () `[private]`

Format a (potentially) multi-line text representation of the contents of the active faults maintained by the `SystemStateMgr` and create a JUCE modal dialog (`AlertWindow`) to display it.

void handleRwyButton (const unsigned *button_index*) `[private]`

Either enable or disable the runway corresponding to `button_index` according to the current button text and force an update of the view.

void handleAudioButton () `[private]`

Toggle the state of the audio system. [Only required for the WTMD prototype system.]

void handleViewLogButton () [private]

Format a (potentially) multi-line text representation of the contents of the fault log maintained by the SystemStateMgr and create a JUCE modal dialog (AlertWindow) to display it.

void enableAudioSystem () [private]

Create and initialize the audio system if the audio file has been specified and the system has not already been created. If successful, change the button text to AUDIO_DISABLE_BUTTON_TEXT and audio status to AUDIO_ENABLED_STATUS_TEXT. If failed, set audio status text to AUDIO_DISABLED_STATUS_TEXT and disable button.

void disableAudioSystem () [private]

If the audio system exists, delete the audio system and change the button text to AUDIO_ENABLE_BUTTON_TEXT and audio status to AUDIO_DISABLED_STATUS_TEXT.

Member Data Documentation

Label* rwy_ids[MAX_RWYS] [private]

Runway name labels in indexible array form.

TextButton* rwy_buttons[MAX_RWYS] [private]

Runway buttons in indexible array form.

Label* rwy_statuses[MAX_RWYS] [private]

Runway status labels in indexible array form.

WakeVAS::Angle asos_direction_ [private]

Wind direction as pulled from last AsosWindsMsg.

WakeVAS::Speed asos_speed_ [private]

Wind speed as pulled from last AsosWindsMsg.

String audio_filename_ [private]

Name of audio file to play for alerts, if configured.

AudioSystem* audio_system_ [private]

If enabled, the audio system component, else NULL.

UpdateTimer* updater_ `[private]`

The JUCE update timer.

Implementation Variable Documentation

const unsigned int TIMER_INTERVAL = 100 `[static]`

Time interval for triggering periodic updates in msec.

const SocketInterface::SocketId COORDINATION_PORT_NUM = 8765 `[static]`

This is used as a quick-and-dirty mutex for audio resources.

static const String ENABLE_BUTTON_TEXT ("Enable") `[static]`

Button text used when button will enable runway.

static const String DISABLE_BUTTON_TEXT ("Disable") `[static]`

Button text used when button will disable runway.

static const String SILENCE_BUTTON_TEXT ("Acknowledge") `[static]`

Button text used when button will acknowledge an alert (effect is to disable runway).

static const String BLANK_BUTTON_TEXT ("") `[static]`

Blank button text.

static const String AVAILABLE_STATUS_TEXT ("Available") `[static]`

Status text to display when runway is available and not enabled.

static const String UNAVAILABLE_STATUS_TEXT ("OFF") `[static]`

Status text to display when runway is unavailable and not enabled.

static const String AVAILABLE_AND_ENABLED_STATUS_TEXT ("WTMD ON") `[static]`

Status text to display when runway is available and enabled.

static const String ENABLED_ONLY_STATUS_TEXT ("ALERT") `[static]`

Status text to display when runway is unavailable but enabled.

static const String BLANK_STATUS_TEXT ("") `[static]`

Status text to display when runway is not configured.

static const String FAILED_SYSTEM_STATE_TEXT ("Fault") `[static]`

Text constant to display when system state is `FAILED`.

static const String INITIALIZING_SYSTEM_STATE_TEXT ("Initializing") `[static]`

Text constant to display when system state is `INITIALIZING`.

static const String OPERATIONAL_SYSTEM_STATE_TEXT ("Operational") `[static]`

Text constant to display when system state is `OPERATIONAL`.

static const String AUDIO_ENABLED_STATUS_TEXT ("Audio alerts enabled") `[static]`

Audio status text to display when audio is enabled.

static const String AUDIO_DISABLED_STATUS_TEXT ("Audio alerts disabled") `[static]`

Audio status text to display when audio is disabled.

static const String AUDIO_ENABLE_BUTTON_TEXT ("Enable Audio") `[static]`

Audio button text to display when audio is disabled.

static const String AUDIO_DISABLE_BUTTON_TEXT ("Disable Audio") `[static]`

Audio button text to display when audio is enabled.

static const Colour amber (192, 128, 0) `[static]`

Color values to use for "amber" which is not in palette of named colors.

6.2.4 SupervisorDialog::UpdateTimer

Public Member Functions

- **UpdateTimer** (**SupervisorDialog** &parent)
- virtual void **timerCallback** ()

Private Attributes

- **SupervisorDialog** & **parent_**

Detailed Description

This class links the JUCE timer to the **`SupervisorDialog`** object in order to invoke the periodic update method.

Constructor & Destructor Documentation

UpdateTimer (SupervisorDialog & *parent*)

Constructor performs default initialization and stores a reference to the **`SupervisorDialog`** for calling into later.

Member Function Documentation

virtual void timerCallback () `[virtual]`

This method reimplementation invokes the periodic update method on the **`SupervisorDialog`** object

---

Member Data Documentation

SupervisorDialog& parent_ `[private]`

The main owner object reference.

6.3 Local Display Process Classes

The main application object for the WTMD local controller display process is `WTMD_LocalApp`. It is responsible for creating all of the other primary application components. `LocalAPPWindow` is the JUCE window class for the local controller display process GUI. The `LocalDialog` class implements the look and behavior of the GUI within the window frame and implements the main periodic processing loop which is invoked through the `LocalDialog::UpdateTimer` class which is a specialization of the JUCE `Timer` class.

6.3.1 WTMD_LocalApp

Public Member Functions

- **WTMD_LocalApp** ()
- virtual **~WTMD_LocalApp** ()

  JUCE Methods

  - virtual void **initialise** (const String &command_line_parms)
  - virtual void **shutdown** ()
  - virtual const String **getApplicationName** ()
  - virtual const String **getApplicationVersion** ()
  - virtual bool **moreThanOneInstanceAllowed** ()
  - virtual void **unhandledException** (const std::exception *e, const String &sourceFile, const int lineNumber)

Private Member Functions

- **WTMD_LocalApp** (const **WTMD_LocalApp** &)
- **WTMD_LocalApp** & **operator=** (const **WTMD_LocalApp** &)

Private Attributes

- String **name_**

  Primary Application Components

  - **WakeVAS::ClientComMgr** * **com_mgr_**
  - **WakeVAS::SystemClock** * **clock_**
  - **WakeVAS::ClientSystemStateMgr** * **state_mgr_**
  - **WakeVAS::ClientRwyStatusMgr** * **rwy_mgr_**
  - **LocalAppWindow** * **window_**

Detailed Description

This is the main application object for the WTMD local controller display process. It is responsible for creating all of the primary application class instances (components) and performing the top-level operations. This class inherits the singleton pattern.

Constructor & Destructor Documentation

WTMD_LocalApp ()

Constructor performs only default initialization of members -- full initialization is performed by **`initialise()`**.

~WTMD_LocalApp () `[virtual]`

Object cleanup is performed in **`shutdown()`**.

WTMD_LocalApp (const WTMD_LocalApp &) `[private]`

Not allowed and not implemented.

Member Function Documentation

void initialise (const String & *command_line_parms*) `[virtual]`

This method parses the command line to extract options, checks for usage violations, initializes stream connections, and then constructs the primary application objects. Then, it creates the application window. The periodic processing is performed by the **`LocalDialog`** which is created within the application window.

void shutdown () `[virtual]`

This method deletes the primary application objects in the reverse order of their creation.

const String getApplicationName () `[virtual]`

Return the name of the application.

const String getApplicationVersion () `[virtual]`

Return the version number of the application.

bool moreThanOneInstanceAllowed () `[virtual]`

Return `true`.

void unhandledException (const std::exception * *e*, const String & *sourceFile*, const int *lineNumber*) `[virtual]`

Default JUCE handler for an unhandled exception. This implementation merely attempts to print the file and line number information.

WTMD_LocalApp& operator= (const WTMD_LocalApp &) `[private]`

Not allowed and not implemented.

Member Data Documentation

String name_ `[private]`

Process name defaults to "Local", but can be overridden by command line options.

WakeVAS::ClientComMgr* com_mgr_ `[private]`

The communications manager.

WakeVAS::SystemClock* clock_ `[private]`

The local time source. Instantiated as a `RealtimeClock`.

WakeVAS::ClientSystemStateMgr* state_mgr_ `[private]`

The system state manager.

WakeVAS::ClientRwyStatusMgr* rwy_mgr_ `[private]`

The runway status manager.

LocalAppWindow* window_ `[private]`

The JUCE application window.

6.3.2 LocalAppWindow

Public Member Functions

- **LocalAppWindow** (const String &title="Local")
- virtual **~LocalAppWindow** ()
- virtual void **closeButtonPressed** ()

Detailed Description

This class specializes the JUCE `DocumentWindow` to customize how it appears, which frame buttons are visible, capture pressing of the close button and to make the window appear on the taskbar.

Constructor & Destructor Documentation

LocalAppWindow (const String & *title* = `"Local"`)

Constructor gets things properly initialized for the way we want the window to appear and behave.

~LocalAppWindow () `[virtual]`

Perform only implicit cleanup.

Member Function Documentation

void closeButtonPressed () `[virtual]`

Handle the pressing of the close button.

6.3.3 LocalDialog

Public Member Functions

- void **paint** (Graphics &g)
- void **resized** ()
- void **buttonClicked** (Button *buttonThatWasClicked)

Private Classes

- class **UpdateTimer**

Private Types

- enum { **MAX_RWYS** = 4 }

Private Member Functions

- void **update** ()
- void **updateView** ()
- void **handleRwyButton** (const unsigned button_index)
- **LocalDialog** (const **LocalDialog** &)
- const **LocalDialog** & **operator=** (const **LocalDialog** &)

Private Attributes

- **WakeVAS::Angle asos_direction_**
- **WakeVAS::Speed asos_speed_**
- **UpdateTimer** * **updater_**
- Label * **state_label_**
- Label * **system_state_**
- Label * **rwy_status1_**
- Label * **rwy_status2_**
- Label * **rwy_status3_**
- Label * **asos_label_**
- Label * **asos_status_**
- TextButton * **rwy_button1_**
- TextButton * **rwy_button2_**
- TextButton * **rwy_button3_**
- Label * **rwy_status4_**
- TextButton * **rwy_button4_**

  Arrays of Per-runway Display Objects

  - std::string **rwy_ids** [MAX_RWYS]
  - TextButton * **rwy_buttons** [MAX_RWYS]
  - Label * **rwy_statuses** [MAX_RWYS]

Implementation Variables

- static const unsigned int **TIMER_INTERVAL** = 100
- static const String **SILENCE_BUTTON_TEXT** ("Acknowledge")
- static const String **BLANK_BUTTON_TEXT** ("")

- static const String **OFF_STATUS_TEXT** ("WTMD OFF")
- static const String **AVAILABLE_AND_ENABLED_STATUS_TEXT** ("WTMD ON")
- static const String **ENABLED_ONLY_STATUS_TEXT** ("ALERT")
- static const String **BLANK_STATUS_TEXT** ("")
- static const String **FAILED_SYSTEM_STATE_TEXT** ("Fault")
- static const String **INITIALIZING_SYSTEM_STATE_TEXT** ("Initializing")
- static const String **OPERATIONAL_SYSTEM_STATE_TEXT** ("Operational")
- static const Colour **amber** (192, 128, 0)

Detailed Description

This is the view and controller portion of the Model-View-Controller pattern used in the WTMD local controller display process. The overall class framework is an auto-generated component, created by the Jucer GUI builder. The non-JUCE code appears within sections of the framework clearly delineated by comments.

Member Function Documentation

void update () `[private]`

This is the head of the periodic processing main loop. It performs the following steps:

1. Tell the `ComMgr` to gather input messages from the core process.
2. Update application components (`SystemStateMgr`, `RwyStatusMgr`). Check the system status to see if a shutdown has been commanded. Look for a new `AsosWindMsg`.
3. Update the view.
4. Tell the `ComMgr` to flush any output messages to the core process.

void updateView () `[private]`

Update the view elements according to the overall WTMD system state.

1. Determine the overall state text to display and set the text and color of the `system_state_` Label accordingly. If the state is not operational, no runway is considered available.
2. For each enabled runway, set the Id, status, button text and button visibility. If the runway is available and enabled, the status text is `AVAILABLE_AND_ENABLED_STATUS_TEXT`, the button text is `BLANK_BUTTON_TEXT` and the button is invisible. If the runway is enabled but not available (alert state), the status text is `ENABLED_ONLY_STATUS_TEXT` the button text is `SILENCE_STATUS_TEXT` and the button is visible. Otherwise, the text is blank and the button is hidden unless none are enabled, in which case the first-line status text is `OFF_STATUS_TEXT`.
3. Update the wind speed and direction labels (which may be hidden if not enabled for display).

void handleRwyButton (const unsigned *button_index*) `[private]`

Disable the runway corresponding to `button_index` if the current button text is `SILENCE_BUTTON_TEXT` and force an update of the view.

Member Data Documentation

std::string rwy_ids[MAX_RWYS] `[private]`

Runway names in indexible array form.

TextButton* rwy_buttons[MAX_RWYS] `[private]`

Runway buttons in indexible array form.

Label* rwy_statuses[MAX_RWYS] `[private]`

Runway status labels in indexible array form.

WakeVAS::Angle asos_direction_ `[private]`

Wind direction as pulled from last `AsosWindsMsg`.

WakeVAS::Speed asos_speed_ `[private]`

Wind speed as pulled from last `AsosWindsMsg`.

UpdateTimer* updater_ `[private]`

The JUCE update timer.

Implementation Variable Documentation

const unsigned int TIMER_INTERVAL = 100 `[static]`

Time interval for triggering periodic updates in msec.

static const String SILENCE_BUTTON_TEXT ("Acknowledge") `[static]`

Button text used when button will acknowledge an alert (effect is to disable runway).

static const String BLANK_BUTTON_TEXT ("") `[static]`

Blank button text.

static const String OFF_STATUS_TEXT ("WTMD OFF") `[static]`

Status text to display when runway is unavailable and not enabled.

static const String AVAILABLE_AND_ENABLED_STATUS_TEXT ("WTMD ON") `[static]`

Status text to display when runway is available and enabled.

static const String ENABLED_ONLY_STATUS_TEXT ("ALERT") `[static]`

Status text to display when runway is unavailable but enabled.

static const String BLANK_STATUS_TEXT ("") `[static]`

Status text to display when runway is not configured.

static const String FAILED_SYSTEM_STATE_TEXT ("Fault") `[static]`

Text constant to display when system state is `FAILED`.

static const String INITIALIZING_SYSTEM_STATE_TEXT ("Initializing") `[static]`

Text constant to display when system state is `INITIALIZING`.

static const String OPERATIONAL_SYSTEM_STATE_TEXT ("Operational") `[static]`

Text constant to display when system state is `OPERATIONAL`.

static const Colour amber (192, 128, 0) `[static]`

Color values to use for "amber" which is not in palette of named colors.

6.3.4 LocalDialog::UpdateTimer

Public Member Functions

- **UpdateTimer** (**LocalDialog** &parent)
- virtual void **timerCallback** ()

Private Attributes

- **LocalDialog** & **parent_**

Detailed Description

This class links the JUCE timer to the **`LocalDialog`** object in order to invoke the periodic update method.

Constructor & Destructor Documentation

UpdateTimer (LocalDialog & *parent*)

Constructor performs default initialization and stores a reference to the **`LocalDialog`** for calling into later.

Member Function Documentation

virtual void timerCallback () `[virtual]`

This method reimplementation invokes the periodic update method on the **`LocalDialog`** object

---

Member Data Documentation

LocalDialog& parent_ `[private]`

The main owner object reference.

6.4 WTMD-IDS Gateway Process Classes

The main application object for the WTMD-IDS gateway process is `WTMD_IDSGwApp`. It is responsible for creating all of the other primary application components and implementing the main periodic processing loop which is invoked through the `GwUpdateTimer` class, which is a specialization of the JUCE `Timer` class.

6.4.1 WTMD_IDSGwApp

Public Member Functions

- **WTMD_IDSGwApp** ()
- virtual **~WTMD_IDSGwApp** ()
- void **update** ()

  JUCE Methods

  - virtual void **initialise** (const String &command_line_parms)
  - virtual void **shutdown** ()
  - virtual const String **getApplicationName** ()
  - virtual const String **getApplicationVersion** ()
  - virtual bool **moreThanOneInstanceAllowed** ()
  - virtual void **unhandledException** (const std::exception *e, const String &sourceFile, const int lineNumber)

Private Member Functions

- **WTMD_IDSGwApp** (const **WTMD_IDSGwApp** &)
- **WTMD_IDSGwApp** & **operator=** (const **WTMD_IDSGwApp** &)

Private Attributes

- **GwUpdateTimer** * **timer_**

  I/O Pieces

  - ACE_TTY_IO * **serial_port_**

  Primary Application Components

  - **WakeVAS::ClientComMgr** * **com_mgr_**
  - **WakeVAS::SystemClock** * **clock_**
  - **WakeVAS::ClientSystemStateMgr** * **state_mgr_**
  - **WakeVAS::ClientRwyStatusMgr** * **rwy_mgr_**

Implementation Variables

- static const unsigned int **TIMER_INTERVAL** = 500

Detailed Description

This is the main application object for the WTMD-IDS gateway process. It is responsible for creating all of the primary application class instances (components) and performing the top-level operations. This class inherits the singleton pattern.

Constructor & Destructor Documentation

WTMD_IDSGwApp ()

Constructor sets the instance pointer and will assert if another instance already exists. Performs only default initialization of members -- full initialization is performed by **`initialise()`**.

~WTMD_IDSGwApp () `[virtual]`

Object cleanup is performed in **`shutdown()`**.

WTMD_IDSGwApp (const WTMD_IDSGwApp &) `[private]`

Not allowed and not implemented.

Member Function Documentation

void initialise (const String & *command_line_parms*) `[virtual]`

This method parses the command line to extract options, checks for usage violations, initializes the serial port, and then constructs the primary application objects. Then, it creates the timer that triggers periodic updates.

void shutdown () `[virtual]`

This method deletes the primary application objects in the reverse order of their creation.

const String getApplicationName () `[virtual]`

Return the name of the application.

const String getApplicationVersion () `[virtual]`

Return the version number of the application.

bool moreThanOneInstanceAllowed () `[virtual]`

Return `true`.

void unhandledException (const std::exception * *e*, const String & *sourceFile*, const int *lineNumber*) `[virtual]`

Default JUCE handler for an unhandled exception. This implementation merely attempts to print the file and line number information.

void update ()

This is the head of the periodic processing main loop. It performs the following steps:

1. Tell the `ComMgr` to gather input messages from the core process.
2. Update application components (`SystemStateMgr`, `RwyStatusMgr`). Check the system status to see if a shutdown has been commanded.
3. Generate a message for transmission based on system and runway state information. Then see if it is time to send the message -- either because the message changed or the inter-output timer expired (20 seconds).
4. Tell the `ComMgr` to flush any output messages to the core process.

WTMD_IDSGwApp& operator= (const WTMD_IDSGwApp &) `[private]`

Not allowed and not implemented.

Member Data Documentation

ACE_TTY_IO* serial_port_ `[private]`

The serial port connection.

WakeVAS::ClientComMgr* com_mgr_ `[private]`

The communications manager.

WakeVAS::SystemClock* clock_ `[private]`

The local time source. Instantiated as a `RealtimeClock`.

WakeVAS::ClientSystemStateMgr* state_mgr_ `[private]`

The system state manager.

WakeVAS::ClientRwyStatusMgr* rwy_mgr_ `[private]`

The runway status manager.

GwUpdateTimer* timer_ `[private]`

The JUCE update timer.

Implementation Variable Documentation

const unsigned int TIMER_INTERVAL = 500 `[static]`

Time interval for triggering periodic updates in msec.

6.4.2 GwUpdateTimer Class Reference

Public Member Functions

- **GwUpdateTimer** (**WTMD_IDSGwApp** &parent)
- virtual void **timerCallback** ()

Private Attributes

- **WTMD_IDSGwApp** & **parent_**

Detailed Description

This class links the JUCE timer to the main application object in order to invoke the periodic update method.

Constructor & Destructor Documentation

GwUpdateTimer (WTMD_IDSGwApp & *parent*)

Constructor performs default initialization and stores a reference to the **`WTMD_IDSGwApp`** for calling into later.

Member Function Documentation

virtual void timerCallback () `[virtual]`

This method reimplementation invokes the periodic update method on the main application object

---

Member Data Documentation

WTMD_IDSGwApp& parent_ `[private]`

The main application object reference.

6.5 Component Classes

This Section describes the classes that make up the underpinnings of the WTMD prototype software. Subsections 6.5.1 through 6.5.5 document classes found in the application context of the prototype. Section 6.5.6 contains the utility classes used to implement the inter-process communications within the prototype context (that is, communications between the processes that comprise the WTMD prototype). Finally, Section 6.5.7 documents the several classes used to represent physical measurements (angles, speeds, and time) in the prototype.

6.5.1 Runway Status

The classes grouped in this subsection are used to represent the WTMD status of a runway and to manage the set of statuses that comprise all of the configured runways. The base manager class is further specialized to accommodate the distributed architecture of the prototype into a version suitable for use within the core process and a version tailored to use in the client processes.

6.5.1.1 RwyStatus

Public Member Functions

- **~RwyStatus** ()
- **RwyStatus** & **operator=** (const **RwyStatus** &rhs)

  Contructors

  - **RwyStatus** ()
  - **RwyStatus** (const std::string &rwy_id)
  - **RwyStatus** (const **RwyStatus** &rhs)

  Comparison methods/operators

  - bool **equal** (const **RwyStatus** &rhs) const
  - bool **operator==** (const **RwyStatus** &rhs) const
  - bool **operator!=** (const **RwyStatus** &rhs) const

  Accessors

  - const std::string & **getRwyId** () const
  - bool **isAvailable** () const
  - bool **isEnabled** () const

  Accessors

  - void **setAvailable** (bool state)
  - void **setEnabled** (bool state)

  Buffer Operations

  - bool **read** (**BinaryDataBuffer** &is)
  - void **write** (**BinaryDataBuffer** &os) const

Private Attributes

- std::string **id_**
- bool **available_**
- bool **enabled_**

Helper Functions

- **BinaryDataBuffer** & **WakeVAS::operator**<< (**BinaryDataBuffer** &data, const **RwyStatus** &rhs)
- **BinaryDataBuffer** & **WakeVAS::operator**>> (**BinaryDataBuffer** &data, **RwyStatus** &rhs)

Detailed Description

The **RwyStatus** class maintains the name and WTMD status of a single runway. It also defines operations useful for moving runway status data to/from a buffer.

Constructor & Destructor Documentation

RwyStatus ()

The default constructor should only be used when the assignment operator or buffer exraction will subsequently be invoked to set the attributes.

RwyStatus (const std::string & *rwy_id*)

Construct a status using the name associated with the runway. The status variables default to `false`.

RwyStatus (const RwyStatus & *rhs*)

Construct a copy of rhs.

~RwyStatus ()

Destructor just implicitly invokes member variable destructors.

Member Function Documentation

RwyStatus & operator= (const RwyStatus & *rhs*)

Make `this` a copy of rhs.

bool operator== (const RwyStatus & *rhs*) const

Return `true` iff all members are equal.

bool operator!= (const RwyStatus & *rhs*) const

Return `true` iff all members are equal.

const std::string& getRwyId () const

Return the name of this runway.

bool isAvailable () const

Return `true` iff this runway is available for WTMD.

bool isEnabled () const

Return `true` iff this runway has been enabled for WTDM.

void setAvailable (bool *state*)

Set the available status.

void setEnabled (bool *state*)

Set the enabled status.

bool read (BinaryDataBuffer & *is*)

Extract value of this from buffer. Returns `true` iff all attributes are read ok.

void write (BinaryDataBuffer & *os*) const

Insert value of this into buffer.

Member Data Documentation

std::string id_ `[private]`

The runway name.

bool available_ `[private]`

Is runway available for WTMD.

bool enabled_ `[private]`

Is runway enabled for WTMD.

Helper Function Documentation

BinaryDataBuffer & WakeVAS::operator<< (BinaryDataBuffer &data, const RwyStatus &rhs)

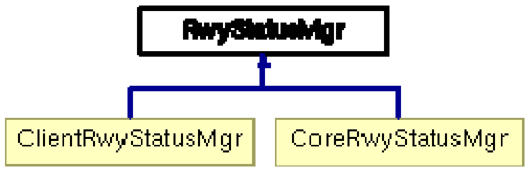
Buffer insertion operator. This will assert on failure.

BinaryDataBuffer & WakeVAS::operator>> (BinaryDataBuffer &data, RwyStatus &rhs)

Buffer extraction operator. This will assert on failure.

6.5.1.2 RwyStatusMgr

Inheritance diagram for RwyStatusMgr:



Public Types

- typedef std::vector< **RwyStatus** > **RwyStatusList**

Public Member Functions

- virtual **~RwyStatusMgr** ()
- size_t **getNumRwys** () const
- const **RwyStatus** & **getRwyStatus** (size_t rwy_index) const
- bool **areAnyEnabled** () const
- bool **areAnyAvailable** () const
- const **RwyStatus** & **getRwyStatus** (const std::string &rwy_id) const
- virtual bool **setAvailable** (size_t rwy_index, bool state)
- bool **setAvailable** (const std::string &rwy_id, bool state)
- virtual bool **setEnabled** (size_t rwy_index, bool state)
- bool **setEnabled** (const std::string &rwy_id, bool state)
- virtual void **update** ()

Static Public Member Functions

- static **RwyStatusMgr** & **getInstance** ()

Protected Member Functions

- **RwyStatusMgr** ()
- size_t **getRwyIndex** (const std::string &rwy_id) const

Protected Attributes

- **RwyStatusList rwy_statuses_**

Private Member Functions

- **RwyStatusMgr** (const **RwyStatusMgr** &rhs)
- **RwyStatusMgr** & **operator=** (const **RwyStatusMgr** &rhs)

Static Private Attributes

- static **RwyStatusMgr** * **instance_** = NULL

Detailed Description

This class defines the interface (and common implementation) for all **RwyStatusMgr** derivatives. Derived classes implement data passing to implement fully distributed behavior. This class implements the singleton pattern as well.

Member Typedef Documentation

typedef std::vector<RwyStatus> RwyStatusList

Type for holding the set of `RunwayStatus` objects.

Constructor & Destructor Documentation

~RwyStatusMgr () `[virtual]`

Destructor implicitly invokes member variable destructors and clears instance pointer.

RwyStatusMgr () `[protected]`

This constructor initializes the singleton instance pointer.

RwyStatusMgr (const RwyStatusMgr & *rhs*) `[private]`

Not allowed and not implemented.

Member Function Documentation

RwyStatusMgr & getInstance () `[static]`

Return the singleton instance. Asserts if instance has not been created.

Reimplemented in **CoreRwyStatusMgr** (*p.67*).size_t getNumRwys () const

Returns the number of `RunwayStatus` objects available.

const RwyStatus & getRwyStatus (size_t *rwy_index*) const

Lookup status info by index (asserts if index >= **getNumRwys()**)

bool areAnyEnabled () const

Return `true` iff at least one runway is in enabled state.

bool areAnyAvailable () const

Return `true` iff at least one runway is in available state.

const RwyStatus& getRwyStatus (const std::string & *rwy_id*) const

Lookup status info by name (asserts if name not found)

bool setAvailable (size_t *rwy_index*, bool *state*) `[virtual]`

Set available attribute and return `true` iff successful. The available status may only be set to `true` if the enabled status if `false`.

Reimplemented in **CoreRwyStatusMgr** (*p.67*).bool setAvailable (const std::string & *rwy_id*, bool *state*)

Helper function to permit specification of runway name instead of index. Will assert if `rwy_id` is not found.

Reimplemented in **CoreRwyStatusMgr** (*p.68*).bool setEnabled (size_t *rwy_index*, bool *state*) `[virtual]`

Set the enabled status attribute and return `true` iff successful.

Reimplemented in **ClientRwyStatusMgr** (*p.70*), and **CoreRwyStatusMgr** (*p.67*).bool setEnabled (const std::string & *rwy_id*, bool *state*)

Helper function to permit specification of runway name instead of index. Will assert if `rwy_id` is not found.

Reimplemented in **CoreRwyStatusMgr** (*p.68*).void update () `[virtual]`

Perform any periodic update work.

Reimplemented in **ClientRwyStatusMgr** (*p.70*), and **CoreRwyStatusMgr** (*p.67*).size_t getRwyIndex (const std::string & *rwy_id*) const `[protected]`

Get index associated with rwy_id. Returns >= **getNumRwys()** if no match

RwyStatusMgr& operator= (const RwyStatusMgr & *rhs*) `[private]`

Not allowed and not implemented.

Member Data Documentation

RwyStatusList rwy_statuses_ `[protected]`

The set of configured `RunwayStatus` objects.

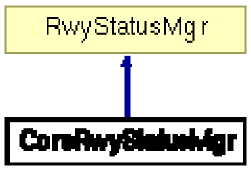
RwyStatusMgr * instance_ = NULL `[static, private]`

The singleton instance

Reimplemented in **CoreRwyStatusMgr** (*p.68*).

6.5.1.3 CoreRwyStatusMgr

Inheritance diagram for CoreRwyStatusMgr:



Public Member Functions

- **CoreRwyStatusMgr** ()
- virtual **~CoreRwyStatusMgr** ()
- void **initialize** (const **RwyStatusList** &rwys)
- virtual void **update** ()
- virtual bool **setAvailable** (size_t rwy_index, bool state)
- virtual bool **setEnabled** (size_t rwy_index, bool state)
- bool **setAvailable** (const std::string &rwy_id, bool state)
- bool **setEnabled** (const std::string &rwy_id, bool state)
- void **reset** ()

Static Public Member Functions

- static **CoreRwyStatusMgr** & **getInstance** ()

Private Member Functions

- **CoreRwyStatusMgr** (const **CoreRwyStatusMgr** &rhs)
- **CoreRwyStatusMgr** & **operator=** (const **CoreRwyStatusMgr** &rhs)

Private Attributes

- **WakeVAS::AbsoluteTime last_msg_tx_**
- bool **need_to_send_**

Static Private Attributes

- static **CoreRwyStatusMgr** * **instance_** = NULL

Implementation Variables

- static const **WakeVAS::RelativeTime TX_INTERVAL** (5., WakeVAS::RelativeTime::SECONDS)

Detailed Description

This class provides an implementation of `RwyStatusMgr` suitable for the core WTMD process. It ensures that changes to runway status are forwarded to the client WTMD processes and processes enable/disable messages received from clients. Like the base class, this class reimplements the singleton pattern to provide access to the additional methods where needed (e.g. **initialize()** and **reset()**).

Constructor & Destructor Documentation

CoreRwyStatusMgr ()

Performs member initialization, but does not initialize the set of runways (see **`initialize()`**).

~CoreRwyStatusMgr () `[virtual]`

Destructor implicitly invokes member variable destructors and clears instance pointer.

CoreRwyStatusMgr (const CoreRwyStatusMgr & *rhs*) `[private]`

Not allowed and not implemented.

Member Function Documentation

CoreRwyStatusMgr & getInstance () `[static]`

Return the singleton instance. Asserts if instance has not been created.

Reimplemented from **RwyStatusMgr** (*p.63*).void initialize (const RwyStatusList & *rwys*)

Initialize the configured set of runways. This is performed once during configuration by the **`WfaWrapper`**, which has access to the necessary configuration data.

void update () `[virtual]`

This implementation of **`update()`** checks for received enable/disable messages and sends a **`RwyStatusMsg`** if the status has changed or if a retransmit interval timer has expired.

Reimplemented from **RwyStatusMgr** (*p.64*).bool setAvailable (size_t *rwy_index*, bool *state*) `[virtual]`

Reimplemantation performs base operation and then updates `need_to_send_`.

Reimplemented from **RwyStatusMgr** (*p.64*).bool setEnabled (size_t *rwy_index*, bool *state*) `[virtual]`

Reimplemantation performs base operation and then updates `need_to_send_`.

Reimplemented from **RwyStatusMgr** (*p.64*).bool setAvailable (const std::string & *rwy_id*, bool *state*)

Helper function to permit specification of runway name instead of index. Will assert if `rwy_id` is not found.

Reimplemented from **RwyStatusMgr** (*p.64*).bool setEnabled (const std::string & *rwy_id*, bool *state*)

Helper function to permit specification of runway name instead of index. Will assert if `rwy_id` is not found.

Reimplemented from **RwyStatusMgr** (*p.64*).void reset ()

This method is needed for cleanup when changing input data during playback. It sets all of the status flags to `false` and resets the interval timer. It does **not** clear the list of runways.

CoreRwyStatusMgr& operator= (const CoreRwyStatusMgr & *rhs*) `[private]`

Not allowed and not implemented.

Member Data Documentation

WakeVAS::AbsoluteTime last_msg_tx_ `[private]`

Record of time of last **`RwyStatusMsg`** transmission for interval timing.

bool need_to_send_ `[private]`

Flag to track that something has changed since last client update.

CoreRwyStatusMgr * instance_ = NULL `[static, private]`

The instance.

Reimplemented from **RwyStatusMgr** (*p.65*).

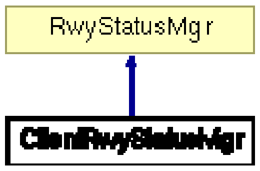
Implementation Variable Documentation

const WakeVAS::RelativeTime TX_INTERVAL(5., WakeVAS::RelativeTime::SECONDS) `[static]`

This constant sets the minimum transmission interval for `RwyStatusMsg` messages.

6.5.1.4 ClientRwyStatusMgr

Inheritance diagram for ClientRwyStatusMgr:



Public Member Functions

- **ClientRwyStatusMgr** ()
- virtual **~ClientRwyStatusMgr** ()
- virtual bool **setEnabled** (size_t rwy_index, bool state)
- virtual void **update** ()

Private Member Functions

- **ClientRwyStatusMgr** (const **ClientRwyStatusMgr** &rhs)
- **ClientRwyStatusMgr** & **operator=** (const **ClientRwyStatusMgr** &rhs)

Detailed Description

This class extends the **`RwyStatusMgr`** to include receiving **`RwyStatusMsg`** messages from the core process and forwarding enable/disable inputs to the core.

Constructor & Destructor Documentation

ClientRwyStatusMgr ()

Pass-through to the base class default constructor.

~ClientRwyStatusMgr () `[virtual]`

Pass-through to the base class destructor.

ClientRwyStatusMgr (const ClientRwyStatusMgr & *rhs*) `[private]`

Not allowed and not implemented.

Member Function Documentation

bool setEnabled (size_t *rwy_index*, bool *state*) `[virtual]`

This class method is reimplemented to generate an **`EnableWtmdMsg`** or **`DisableWtmdMsg`** and send it to the core process for processing. If the core heartbeat is not being received and the command is to disable, the action is also performed locally so that an alert state can be cleared locally (alert state is defined as a runway status of enabled but not available or a status of enabled when the system state is not operational).

Reimplemented from **RwyStatusMgr** (*p.64*).void update () `[virtual]`

In addition to base class processing, this reimplementation looks for **`RwyStatusMsg`** messages from the core and uses them to update the the information held.

Reimplemented from **RwyStatusMgr** (*p.64*).ClientRwyStatusMgr& operator= (const ClientRwyStatusMgr & *rhs*) `[private]`

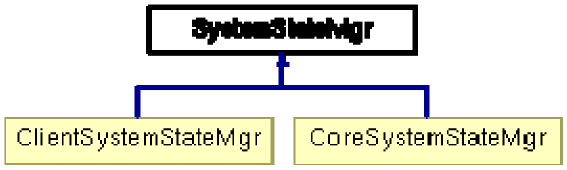
Not allowed and not implemented.

6.5.2 System State Manager

The classes grouped in this subsection are used to represent the operating state of the WTMD system. The base manager class is further specialized to accommodate the distributed architecture of the prototype into a version suitable for use within the core process and a version tailored for use in the client processes. Additional classes/structures are used to represent fault information and the data about connected client processes that the core needs to retain.

6.5.2.1 SystemStateMgr

Inheritance diagram for SystemStateMgr:



Public Classes

- class **Fault**

Public Types

- typedef std::list< **Fault** > **FaultList**
- enum **State** { **FAILED** = 0, **INITIALIZING** = 1, **OPERATIONAL** = 2, **SHUTDOWN** = 99 }

Public Member Functions

- virtual **~SystemStateMgr** ()
- virtual void **update** ()

  Accessors

  - const **FaultList** & **getFaultList** () const
  - const **FaultList** & **getFaultLog** () const
  - **State getState** () const

  Mutators

  - virtual **State putState** (**State** next_state)
  - virtual void **addFault** (const std::string &fault)
  - virtual void **removeFault** (const std::string &fault, **State** next_state=OPERATIONAL)

Static Public Member Functions

- static **SystemStateMgr** & **getInstance** ()

Protected Member Functions

- **SystemStateMgr** ()

- void **putData** (**State** state, const **FaultList** &faults)
- void **putLogData** (const **FaultList** &fault_log)
- virtual void **addLogItem** (const **Fault** &flt)

Static Protected Attributes

- static const std::string **CORE_TIMEOUT_FAULT**

Private Member Functions

- **SystemStateMgr** (const **SystemStateMgr** &)
- **SystemStateMgr** & **operator=** (const **SystemStateMgr** &)

Private Attributes

- **State system_state_**
- **FaultList faults_**
- **FaultList fault_log_**

Static Private Attributes

- static **SystemStateMgr** * **instance_** = NULL

Implementation Variables

- static const size_t **MAX_LOG_FAULTS** = 20

Detailed Description

This class provides an abstract interface for accessing the state of the overall system, including getting active fault codes and a list of previous faults which have now been resolved. This class also implements the singleton pattern. The specific implementations, **`CoreSystemStateMgr`** and **`ClientSystemStateMgr`**, implement tracking of the health of the other systems. The `CoreSSM` monitors the presence and health of the clients, and the `ClientSSMs` make sure they are receiving information from the core process. The `CoreSSM` is also responsible for detecting the presence of new and dead clients and adding/removing them from the **`CoreComMgr`**.

Member Typedef Documentation

typedef std::list<Fault> FaultList

Type for holding the set of **`Fault`** objects.

Member Enumeration Documentation

enum State

Enumeration used to summarize the state of the system.

Enumerator:

FAILED At least one fault is active.

INITIALIZING The system has not received enough wind data yet to go operational.

OPERATIONAL The system is in the normal operating state.

SHUTDOWN The system has been shutdown.

Constructor & Destructor Documentation

~SystemStateMgr () `[virtual]`

Destructor implicitly invokes member variable destructors and clears instance pointer.

SystemStateMgr () `[protected]`

Perform basic initialization and set state summary to `INITIALIZING`. Also sets the instance pointer and will assert if another instance already exists.

SystemStateMgr (const SystemStateMgr &) `[private]`

Not allowed and not implemented.

Member Function Documentation

SystemStateMgr & getInstance () `[static]`

Return the singleton instance. Asserts if instance has not been created.

const FaultList& getFaultList () const

Return set of active faults.

State getState () const

Return set of previous faults. Return the current system state summary enumeration.

SystemStateMgr::State putState (State *next_state*) `[virtual]`

Update the state to next_state if possible. If active faults are present, will only allow state change to `FAILED` or `SHUTDOWN`. Returns the resulting state, either `next_state` or `FAILED`.

void addFault (const std::string & *fault*) `[virtual]`

Adds `fault` to the set of active faults (if not already present). Automatically calls **`putState()`** to set state to `FAILED`.

Reimplemented in **ClientSystemStateMgr** (*p.85*).void removeFault (const std::string & *fault*, State *next_state* = `OPERATIONAL`) `[virtual]`

Remove indicated fault from the list of active faults. If `fault` is removed, the end time is noted and the fault is added to the fault log. If no faults are active after removal, **`putState()`** is called to set the state to **`next_state`**.

Parameters:

fault Fault to remove (match based on fault text).

next_state this will be the new state if no other faults

void update () `[virtual]`

Perform any periodic update work.

Reimplemented in **ClientSystemStateMgr** (*p.85*), and **CoreSystemStateMgr** (*p.80*).void putData (State *state*, const FaultList & *faults*) `[protected]`

Replace the current state and fault data with that supplied in the arguments, whiping out any prior contents. This is typically used when receiving such data in a message from a central server (core process) or as a means of resetting the data.

void putLogData (const FaultList & *fault_log*) `[protected]`

Replace the current fault log data with that supplied in the arguments, whiping out any prior contents. This is typically used when receiving such data in a message from a central server (core process) or as a means of resetting the data.

void addLogItem (const Fault & *flt*) `[protected, virtual]`

Add a fault to the log. This operation is performed when clearing a fault from the active fault list or when a fault log item is received via a message.

SystemStateMgr& operator= (const SystemStateMgr &) `[private]`

Not allowed and not implemented.

Member Data Documentation

const std::string CORE_TIMEOUT_FAULT `[static, protected]`

Standard string used to indicate client is not hearing core

State system_state_ `[private]`

The current system state summary enumeration.

FaultList faults_ `[private]`

The set of active faults.

FaultList fault_log_ `[private]`

The set of previous faults, with most recently cleared first.

SystemStateMgr * instance_ = NULL `[static, private]`

The singleton instance.

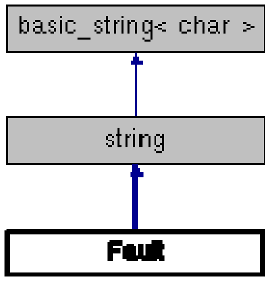
Implementation Variable Documentation

const size_t MAX_LOG_FAULTS = 20 `[static]`

Maximum number of faults to keep in fault log.

6.5.2.2 SystemStateMgr::Fault

Inheritance diagram for SystemStateMgr::Fault:



Accessors

- const **WakeVAS::AbsoluteTime** & **getStartTime** () const
- const **WakeVAS::AbsoluteTime** & **getEndTime** () const
- void **print** (std::ostream &os) const
- bool **read** (**BinaryDataBuffer** &buff)
- void **write** (**BinaryDataBuffer** &buff) const
- **WakeVAS::AbsoluteTime start_time_**
- **WakeVAS::AbsoluteTime end_time_**

Public Member Functions

- void **noteEndTime** ()

  Contructors

  - **Fault** ()
  - **Fault** (const std::string &text)

Helper Functions

- std::ostream & **WakeVAS::operator<<** (std::ostream &os, const SystemStateMgr::Fault &flt)
- BinaryDataBuffer & **WakeVAS::operator<<** (BinaryDataBuffer &buff, const SystemStateMgr::Fault &flt)
- BinaryDataBuffer & **WakeVAS::operator>>** (BinaryDataBuffer &buff, SystemStateMgr::Fault &flt)

Detailed Description

The **Fault** class keeps track of the name of a fault as well as when it begins and when it clears.

Constructor & Destructor Documentation

Fault ()

The default constructor should only be used when the assignment operator or buffer exraction will subsequently be invoked to set the attributes.

Fault (const std::string & *text*)

This constructor initializes start time to current system time.

Member Function Documentation

void noteEndTime ()

This method is called when a fault has cleared to capture the system time at which it cleared.

const WakeVAS::AbsoluteTime& getStartTime () const

Return time at which fault was first noted.

const WakeVAS::AbsoluteTime& getEndTime () const

Return time at which fault was cleared.

void print (std::ostream & *os*) const

Write textual description of fault to stream including start time and end time if it has been noted.

bool read (BinaryDataBuffer & *buff*)

Extract value of this from buffer. Returns `true` iff all attributes are read ok.

void write (BinaryDataBuffer & *buff*) const

Insert value of this into buffer.

Member Data Documentation

WakeVAS::AbsoluteTime start_time_ `[private]`

Time at which fault was first noted.

WakeVAS::AbsoluteTime end_time_ `[private]`

Time at which fault was cleared.

Helper Function Documentation

std::ostream & **WakeVAS::operator<<** (std::ostream &os, const SystemStateMgr::Fault &flt)

Buffer insertion operator for printing a **`Fault`** to a **`stream`**.

BinaryDataBuffer & **WakeVAS::operator<<** (BinaryDataBuffer &buff, const SystemStateMgr::Fault &flt)

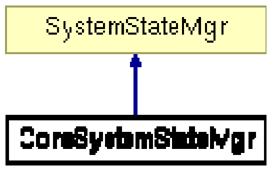
Buffer insertion operator for writing a **`Fault`** to a **`BinaryDataBuffer`**.

BinaryDataBuffer & **WakeVAS::operator>>** (BinaryDataBuffer &buff, SystemStateMgr::Fault &flt)

Buffer insertion operator for reading a **`Fault`** from a **`BinaryDataBuffer`**.

6.5.2.3 CoreSystemStateMgr

Inheritance diagram for CoreSystemStateMgr:



Public Member Functions

- **CoreSystemStateMgr** ()
- virtual **~CoreSystemStateMgr** ()
- virtual **State putState** (**State** next_state)
- virtual void **update** ()
- void **reset** ()

Protected Classes

- struct **ClientData**

Protected Types

- typedef std::map< std::string, **ClientData** > **ClientSet**

Protected Member Functions

- void **updateClient** (const **ClientStatusMsg** &msg)
- void **sendHeartbeat** (const **WakeVAS::AbsoluteTime** &now)
- virtual void **addLogItem** (const Fault &flt)

Private Member Functions

- **CoreSystemStateMgr** (const **CoreSystemStateMgr** &)
- **CoreSystemStateMgr** & **operator=** (const **CoreSystemStateMgr** &)

Private Attributes

- **ClientSet clients_**
- **WakeVAS::AbsoluteTime last_heartbeat_tx_**

Helper Functions

- static const std::string & **makeUniqueName** (const std::string &name, const std::string &host, int port)

Implementation Variables

- static const **RelativeTime HEARTBEAT_INTERVAL** (.5, RelativeTime::SECONDS)
- static const **RelativeTime CLIENT_HEARTBEAT_TIMEOUT** (12., RelativeTime::SECONDS)

- static const **RelativeTime CLIENT_REMOVE_TIMEOUT** (30., RelativeTime::SECONDS)

Detailed Description

This class provides an implementation of **`SystemStateMgr`** suitable for the core WTMD process. It ensures that changes to the system status are forwarded to the client WTMD processes and processes status messages received from clients, including performing maintenance of client connections.

Member Typedef Documentation

typedef std::map<std::string, ClientData> ClientSet `[protected]`

Type for holding the set of **`ClientData`** objects.

Constructor & Destructor Documentation

CoreSystemStateMgr ()

Perform basic initialization and set `last_heartbeat_tx_` to a time that will force the sending of a message the first time **`update()`** is called.

~CoreSystemStateMgr () `[virtual]`

Perform basic cleanup.

CoreSystemStateMgr (const CoreSystemStateMgr &) `[private]`

Not allowed and not implemented.

Member Function Documentation

SystemStateMgr::State putState (State *next_state*) `[virtual]`

Perform the base class **`putState()`** and send a new status message if the system state changes.

void update () `[virtual]`

Process client status messages from clients (adding new clients if appropriate), remove expired clients, and send status message if data has changed or if heartbeat interval has expired since last transmission.

Reimplemented from **SystemStateMgr** (*p.74*).void reset ()

Clear log and faults, but not client list. Sets status to `INITIALIZING`.

void updateClient (const ClientStatusMsg & *msg*) `[protected]`

Process the status message received from a client.

1. Add a record for this client if this is the first received.
2. Use the failure string to add/remove/modify the fault associated with this client (a core timeout is ignored from a relatively new (20 seconds) client).
3. Update last update time for the client.

void sendHeartbeat (const WakeVAS::AbsoluteTime & *now*) `[protected]`

Sends state data as heartbeat and note tx time.

void addLogItem (const Fault & *flt*) `[protected, virtual]`

Extends the base class implementation to forward item to clients.

CoreSystemStateMgr& operator= (const CoreSystemStateMgr &) `[private]`

Not allowed and not implemented.

Member Data Documentation

ClientSet clients_ `[private]`

The set of clients known to the core process.

WakeVAS::AbsoluteTime last_heartbeat_tx_ `[private]`

The last time a heartbeat message was sent.

Helper Function Documentation

static const std::string& makeUniqueName (const std::string & *name*, const std::string & *host*, int *port*) `[static]`

Helper function for converting the client name, host, and port number into a text string that should be unique accross all clients (since two clients cannot both be receiving messages from the same port number on the same host).

Implementation Variable Documentation

const RelativeTime HEARTBEAT_INTERVAL(.5, RelativeTime::SECONDS) `[static]`

How often heartbeat messages (`CoreStatusMsg`) are transmitted.

const RelativeTime CLIENT_HEARTBEAT_TIMEOUT(12., RelativeTime::SECONDS) `[static]`

How long to wait for a client heartbeat messages (`ClientStatusMsg`) before considering the client to be late.

const RelativeTime CLIENT_REMOVE_TIMEOUT(30., RelativeTime::SECONDS) `[static]`

How long to wait for a client heartbeat messages (`ClientStatusMsg`) before removing the client and stopping output to it.

6.5.2.4 CoreSystemStateMgr::ClientData

Public Member Functions

- **ClientData** ()

Public Attributes

- **State state_**
- std::string **fault_**
- **WakeVAS::AbsoluteTime first_joined_**
- **WakeVAS::AbsoluteTime last_update_**

Detailed Description

The **`ClientData`** structure is used to keep track of information about each of the connected clients

---

Constructor & Destructor Documentation

ClientData ()

There is no default constructor for **`AbsoluteTime`**, so must explicitly construct.

Member Data Documentation

State state_

State summary reported by client.

std::string fault_

Fault (if any) reported by client.

WakeVAS::AbsoluteTime first_joined_

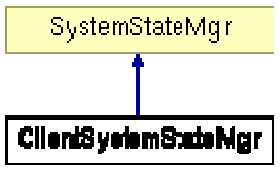
Time at which first status message was received from client.

WakeVAS::AbsoluteTime last_update_

Time at which most recent status message was received from client.

6.5.2.5 ClientSystemStateMgr

Inheritance diagram for ClientSystemStateMgr:



Public Member Functions

- **ClientSystemStateMgr** (const std::string &my_name, const std::string &my_host, int receive_port)
- virtual **~ClientSystemStateMgr** ()
- virtual void **addFault** (const std::string &fault)
- virtual void **removeFault** (const std::string &fault, **State** next_state=OPERATIONAL)
- virtual **State putState** (**State** next_state)
- virtual void **update** ()

Protected Member Functions

- void **sendHeartbeat** (const **WakeVAS::AbsoluteTime** &now)

Private Member Functions

- **ClientSystemStateMgr** ()
- **ClientSystemStateMgr** (const **ClientSystemStateMgr** &)
- **ClientSystemStateMgr** & **operator=** (const **ClientSystemStateMgr** &)

Private Attributes

- std::string **name_**
- std::string **host_**
- int **receive_port_**
- **FaultList local_faults_**
- **WakeVAS::AbsoluteTime last_heartbeat_tx_**
- **WakeVAS::AbsoluteTime last_heartbeat_rx_**

Implementation Variables

- static const **RelativeTime HEARTBEAT_INTERVAL** (5., RelativeTime::SECONDS)
- static const **RelativeTime CORE_HEARTBEAT_TIMEOUT** (12., RelativeTime::SECONDS)

Detailed Description

This class extends the `SystemStateMgr` to include receiving `CoreStatusMsg` messages from the core process and forwarding **ClientStatusMsg** inputs to the core. It also maintains locally generated faults (which are also forwarded to the core) and detects loss of heartbeat messages from the core.

Constructor & Destructor Documentation

ClientSystemStateMgr (const std::string & *my_name*, const std::string & *my_host*, int *receive_port*)

The constructor initializes the stored name, host, and port values. The application is assumed to use the **`ComMgr`** to actually open the port. It also initializes the transmit timer so that a heartbeat message will be generated on the next call to **`update()`** and the receive timer as if a **CoreStatusMsg** had just been received.

~ClientSystemStateMgr () `[virtual]`

Destructor just performs routine cleanup.

ClientSystemStateMgr () `[private]`

Not allowed and not implemented.

ClientSystemStateMgr (const ClientSystemStateMgr &) `[private]`

Not allowed and not implemented.

Member Function Documentation

void addFault (const std::string & *fault*) `[virtual]`

This method extends the base class implementation by also keeping a list of locally generated faults to avoid having them be lost when processing updates from the core. The fault will not be forwarded to the core until **`update()`** is called.

Reimplemented from **SystemStateMgr** (*p.74*).void removeFault (const std::string & *fault*, State *next_state* = `OPERATIONAL`) `[virtual]`

This method extends the base class implementation by also removing the fault from the list of locally generated faults. The core will remove the fault when it receives the next **ClientStatusMsg** from this client without this fault.

SystemStateMgr::State putState (State *next_state*) `[virtual]`

This method extends the base class implementation to send a heartbeat message (**`ClientStatusMsg`**) if the state actually changes.

void update () `[virtual]`

Perform periodic processing.

1. Process the status and fault log messages received from a core. When updating the active fault list, local faults must be added to faults reported by the core (duplicates will be ignored).
2. Check time since last core heartbeat reception and add/remove core heartbeat timeout fault as appropriate.
3. Call **`sendHeartbeat()`** if state or fault list has changed or if the heartbeat interval has expired.

Reimplemented from **SystemStateMgr** (*p.74*).void sendHeartbeat (const WakeVAS::AbsoluteTime & *now*) `[protected]`

Generate a **`ClientStatusMsg`** (heartbeat) and send it to the core. The failure string in the message is set to the first fault in the local list or an emtpy string if the local fault list is empty. The time of transmission is noted for interval timing.

ClientSystemStateMgr& operator= (const ClientSystemStateMgr &) `[private]`

Not allowed and not implemented.

Member Data Documentation

std::string name_ `[private]`

Process name used in status message.

std::string host_ `[private]`

Host name used in status message.

int receive_port_ `[private]`

Port number used in status message.

FaultList local_faults_ `[private]`

Set of locally generated faults.

WakeVAS::AbsoluteTime last_heartbeat_tx_ `[private]`

The last time a heartbeat message was sent.

WakeVAS::AbsoluteTime last_heartbeat_rx_ `[private]`

Time at which most recent status message was received from core.

Implementation Variable Documentation

const RelativeTime HEARTBEAT_INTERVAL(5., RelativeTime::SECONDS) `[static]`

How often heartbeat messages (`ClientStatusMsg`) are transmitted.

const RelativeTime CORE_HEARTBEAT_TIMEOUT(12., RelativeTime::SECONDS) `[static]`

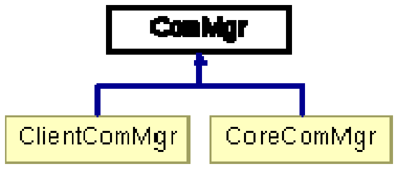
How long to wait for a core heartbeat messages (`CoreStatusMsg`) before considering the core to be late.

6.5.3 Communication Manager

The classes grouped in this subsection are used to represent a generic inter-process message passing facility for inter-process communications within the WTMD system. The base manager class is further specialized to accommodate the special needs of the core and the client processes

6.5.3.1 ComMgr

Inheritance diagram for ComMgr:



Public Types

- typedef std::vector< **Msg** * > **MsgList**

Public Member Functions

- virtual **~ComMgr** ()
- virtual const **MsgList** & **getReceivedData** () const =0
- virtual void **send** (const **Msg** &data)=0

  - virtual void **gatherInput** ()
  - virtual void **flushOutput** ()
  - virtual void **reset** ()

Static Public Member Functions

- static **ComMgr** & **getInstance** ()

Protected Member Functions

- **ComMgr** ()

Private Member Functions

- **ComMgr** (const **ComMgr** &)
- **ComMgr** & **operator=** (const **ComMgr** &)

Static Private Attributes

- static **ComMgr** * **instance_** = NULL

Detailed Description

This class provides an abstract interface for sending and receiving communications. This base class implements the singleton pattern, but behavior implementation is provided by either a **`CoreComMgr`** or a **`ClientComMgr`**.

Member Typedef Documentation

typedef std::vector<Msg*> MsgList

Data type for holding a set of messages.

Constructor & Destructor Documentation

~ComMgr () `[virtual]`

Base class destructor cleans up the singleton instance.

ComMgr () `[protected]`

Only derived classes can construct the abstract base class. Initializes the instance pointer. Asserts if an instance already exists

ComMgr (const ComMgr &) `[private]`

Not allowed and not implemented.

Member Function Documentation

ComMgr & getInstance () `[static]`

Returns the singleton instance. Asserts if instance has not been created.

Reimplemented in **CoreComMgr** (*p.95*).virtual const MsgList& getReceivedData () const `[pure virtual]`

Return set of received messages.

Implemented in **ClientComMgr** (*p.92*), and **CoreComMgr** (*p.95*).virtual void send (const Msg & *data*) `[pure virtual]`

Output the supplied message.

Implemented in **ClientComMgr** (*p.92*), and **CoreComMgr** (*p.96*).void gatherInput () `[virtual]`

Discard any prior gathered messages and gather new waiting input messages.

Reimplemented in **ClientComMgr** (*p.92*), and **CoreComMgr** (*p.96*).void flushOutput () `[virtual]`

Flush output buffers (if any).

Reimplemented in **ClientComMgr** (*p.92*), and **CoreComMgr** (*p.96*).void reset () `[virtual]`

Discard any buffered input or output. Does not affect any open connections.

Reimplemented in **ClientComMgr** (*p.93*), and **CoreComMgr** (*p.96*).ComMgr& operator= (const ComMgr &) `[private]`

Not allowed and not implemented.

Member Data Documentation

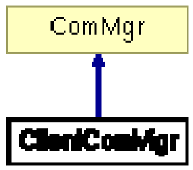
ComMgr * instance_ = NULL `[static, private]`

The instance.

Reimplemented in **CoreComMgr** (*p.97*).

6.5.3.2 ClientComMgr

Inheritance diagram for ClientComMgr:



Public Member Functions

- **ClientComMgr** ()
- virtual **~ClientComMgr** ()
- bool **initialize** (int local_receive_port_num, const std::string &core_hostname, int core_receive_port_num)
- virtual const **MsgList** & **getReceivedData** () const
- virtual void **send** (const **Msg** &data)
- virtual void **gatherInput** ()
- virtual void **flushOutput** ()
- virtual void **reset** ()

Protected Member Functions

- void **clearMsgs** ()

Private Member Functions

- **ClientComMgr** (const **ClientComMgr** &)
- **ClientComMgr** & **operator=** (const **ClientComMgr** &)

Private Attributes

- **WakeVAS::SocketInterface::SocketId receive_socket_**
- **MsgList received_msgs_**
- **WakeVAS::SocketInterface::SocketId send_socket_**

Detailed Description

This class provides an implementation of **ComMgr** suitable for client WTMD processes. It maintains an input socket for receiving messages and and output socket for sending messages to the core process.

Constructor & Destructor Documentation

ClientComMgr ()

Initializes the member variables, but does not open any connections/sockets. Asserts if a **ComMgr** instance already exists.

~ClientComMgr () `[virtual]`

Closes all connections and cleans up.

ClientComMgr (const ClientComMgr &) `[private]`

Not allowed and not implemented.

Member Function Documentation

bool initialize (int *local_receive_port_num*, const std::string & *core_hostname*, int *core_receive_port_num*)

Port number core is using to receive input.

Open the UDP socket used for receiving messages and the UDP socket used for sending messages to the core. Returns `true` iff succeeds in openning both sockets.

Parameters:

local_receive_port_num Port number to open for receiving input.

core_hostname Host name or #.#.#.# IP address where core is running.

const ComMgr::MsgList & getReceivedData () const `[virtual]`

Return set of received messages.

Implements **ComMgr** (*p.89*).void send (const Msg & *data*) `[virtual]`

Output the supplied message.

Implements **ComMgr** (*p.89*).void gatherInput () `[virtual]`

Discard any prior gathered messages and gather new waiting input messages.

Reimplemented from **ComMgr** (*p.90*).void flushOutput () `[virtual]`

Flush output buffers (if any).

Reimplemented from **ComMgr** (*p.90*).void reset () `[virtual]`

Discard any buffered input or output. Does not affect any open connections.

Reimplemented from **ComMgr** (*p.90*).void clearMsgs () `[protected]`

Discard the gathered input messages.

ClientComMgr& operator= (const ClientComMgr &) `[private]`

Not allowed and not implemented.

Member Data Documentation

WakeVAS::SocketInterface::SocketId receive_socket_ `[private]`

Socket used to receive messages.

MsgList received_msgs_ `[private]`

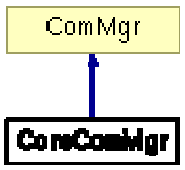
Set of messages gathered in last call to **`gatherInput()`**.

WakeVAS::SocketInterface::SocketId send_socket_ `[private]`

Socket used to send messages to the core.

6.5.3.3 CoreComMgr

Inheritance diagram for CoreComMgr:



Public Member Functions

- **CoreComMgr** ()
- virtual **~CoreComMgr** ()
- bool **initialize** (int port_num)
- virtual const **MsgList** & **getReceivedData** () const
- virtual void **send** (const **Msg** &data)
- virtual void **gatherInput** ()
- virtual void **flushOutput** ()
- virtual void **reset** ()

  Client Management Methods

  - void **addClient** (const std::string &name, const std::string &hostname, int port_num)
  - void **deleteClient** (const std::string &name)

Static Public Member Functions

- static **CoreComMgr** & **getInstance** ()

Protected Member Functions

- void **clearMsgs** ()

Private Types

- typedef std::map< std::string, **WakeVAS::SocketInterface::SocketId** > **ClientSet**

Private Member Functions

- **CoreComMgr** (const **CoreComMgr** &)
- **CoreComMgr** & **operator=** (const **CoreComMgr** &)

Private Attributes

- **WakeVAS::SocketInterface::SocketId receive_socket_**
- **MsgList received_msgs_**
- **ClientSet clients_**

Static Private Attributes

- static **CoreComMgr** * **instance_** = NULL

Detailed Description

This class provides an implementation of **ComMgr** suitable for the core WTMD process. It maintains connections with the client WTMD processes and distributes sent messages to each client. Like the base class, this class reimplements the singleton pattern to provide access to the additional methods where needed.

Member Typedef Documentation

typedef std::map<std::string, WakeVAS::SocketInterface::SocketId> ClientSet `[private]`

Set for holding connected client information assciated with client name.

Constructor & Destructor Documentation

CoreComMgr ()

Initializes the instance pointer, but does not open any connections/sockets. Asserts if an instance already exists.

~CoreComMgr () `[virtual]`

Closes all connections and cleans up, including clearing the instance pointer.

CoreComMgr (const CoreComMgr &) `[private]`

Not allowed and not implemented.

Member Function Documentation

CoreComMgr & getInstance () `[static]`

Returns the singleton instance. Asserts if instance has not been created.

Reimplemented from **ComMgr** (*p.89*).bool initialize (int *port_num*)

Open the UDP socket used for receiving messages. Returns `true` iff succeeds in openning the socket.

const ComMgr::MsgList & getReceivedData () const `[virtual]`

Return set of received messages.

Implements **ComMgr** (*p.89*).void send (const Msg & *data*) `[virtual]`

Output the supplied message to each connected client.

Implements **ComMgr** (*p.89*).void gatherInput () `[virtual]`

Discard any prior gathered messages and gather new waiting input messages.

Reimplemented from **ComMgr** (*p.90*).void flushOutput () `[virtual]`

Flush output buffers (if any).

Reimplemented from **ComMgr** (*p.90*).void reset () `[virtual]`

Discard any buffered input or output. Does not affect any open connections.

Reimplemented from **ComMgr** (*p.90*).void addClient (const std::string & *name*, const std::string & *hostname*, int *port_num*)

Port number client is using to receive messages.

Open an output UDP socket to talk to the named client using the supplied host/address and port number.

Parameters:

name Name to associate with this client, must be unique.

hostname Host name or #.#.#.# IP address where client is running.

void deleteClient (const std::string & *name*)

name matches that used in **addClient()**.

Remove the named client and close the associated socket

void clearMsgs () `[protected]`

Discard the gathered input messages.

CoreComMgr& operator= (const CoreComMgr &) `[private]`

Not allowed and not implemented.

Member Data Documentation

WakeVAS::SocketInterface::SocketId receive_socket_ `[private]`

Socket used to receive messages.

MsgList received_msgs_ `[private]`

Set of messages gathered in last call to **`gatherInput()`**.

ClientSet clients_ `[private]`

Set of connected clients.

CoreComMgr * instance_ = NULL `[static, private]`

The instance.

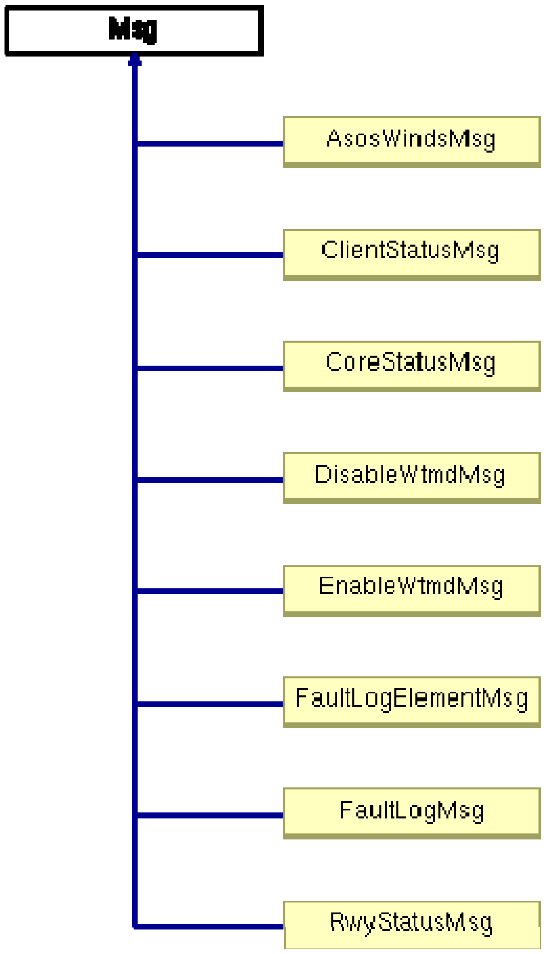
Reimplemented from **ComMgr** (*p.90*).

6.5.4 Messages

This subsection documents the set of classes used to represent the information passed between the processes of the WTMD prototype system. Each type of information that can be passed is represented by a class that encapsulates the details of how the information is formatted into an array of bytes for transmission.

6.5.4.1 Msg

Inheritance diagram for Msg:



Public Types

- enum **Type** { **CORE_STATUS** = 10, **FAULT_LOG** = 11, **FAULT_LOG_ELEMENT** = 12, **CLIENT_STATUS** = 20, **RWY_STATUS** = 30, **ASOS_WINDS** = 31, **ENABLE_WTMD** = 40, **DISABLE_WTMD** = 41 }

Public Member Functions

- virtual **~Msg** ()

- virtual **Msg** * **duplicate** () const =0
- virtual bool **read** (**BinaryDataBuffer** &is)
- virtual void **write** (**BinaryDataBuffer** &os) const

Accessors

- **Type getType** () const

Static Public Member Functions

- static **Msg** * **extract** (**BinaryDataBuffer** &data)

Protected Member Functions

Contructors

- **Msg** (**Type** type)
- **Msg** (const **Msg** &rhs)

Private Member Functions

- **Msg** ()

Private Attributes

- **Type msg_type_**

Detailed Description

Pure virtual base class for all WTMD message classes. This class defines the message type and the virtual methods for message duplication and reading/writing messages to/from a **`BinaryDataBuffer`**. Also defines the static method for reconstructing a message from a buffer.

Member Enumeration Documentation

enum Type

Enumeration used for identifying type of message. It is useful when writing `switch()` statements to handle messages. This value is also placed in a buffer first when inserting a **Msg** into a buffer.

Constructor & Destructor Documentation

~Msg () `[virtual]`

Base class destructor is declared virtual to ensure derived class destructor is called when deleted through a base class pointer.

Msg (Type *type*) `[protected]`

Only derived classes can construct virtual base class.

Msg (const Msg & *rhs*) `[protected]`

Standard copy constructor. Again, only derived classes can construct virtual base class.

Msg () `[private]`

Not allowed and not implemented. **`Msg`** may only be constructed by specifying the `Type`.

Member Function Documentation

virtual Msg* duplicate () const `[pure virtual]`

Return an exact copy of self allocated on heap.

Implemented in **CoreStatusMsg** (*p.109*), **FaultLogMsg** (*p.114*), **FaultLogElementMsg** (*p.116*), **ClientStatusMsg** (*p.111*), **RwyStatusMsg** (*p.103*), **EnableWtmdMsg** (*p.105*), **DisableWtmdMsg** (*p.107*), and **AsosWindsMsg** (*p.118*).bool read (BinaryDataBuffer & *is*) `[virtual]`

Extract **Msg** contents from a buffer. Derived classes should call parent class **read()** before removing data. Returns `true` iff extraction OK.

Reimplemented in **CoreStatusMsg** (*p.109*), **FaultLogMsg** (*p.114*), **FaultLogElementMsg** (*p.116*), **ClientStatusMsg** (*p.111*), **RwyStatusMsg** (*p.103*), **EnableWtmdMsg** (*p.105*), **DisableWtmdMsg** (*p.107*), and **AsosWindsMsg** (*p.118*).void write (BinaryDataBuffer & *os*) const `[virtual]`

Insert **Msg** contents into a buffer. Derived classes should call parent class **write()** before adding data.

Reimplemented in **CoreStatusMsg** (*p.109*), **FaultLogMsg** (*p.114*), **FaultLogElementMsg** (*p.116*), **ClientStatusMsg** (*p.111*), **RwyStatusMsg** (*p.103*), **EnableWtmdMsg** (*p.105*), **DisableWtmdMsg** (*p.107*), and **AsosWindsMsg** (*p.118*).Type getType () const

Returns the type enumeration.

Msg * extract (BinaryDataBuffer & *data*) `[static]`

Static method for reconstituting a derived **Msg** class from data in a buffer. Returned pointer, if non-NULL, is allocated from heap and should eventually be deleted.

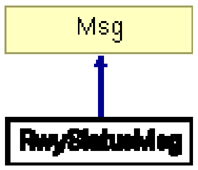
Member Data Documentation

Type msg_type_ `[private]`

The message type passed by derived class during construction.

6.5.4.2 RwyStatusMsg

Inheritance diagram for RwyStatusMsg:



Public Types

- typedef **RwyStatusMgr::RwyStatusList RwyStatusList**

Public Member Functions

- virtual **~RwyStatusMsg** ()
- virtual **Msg** * **duplicate** () const
- virtual bool **read** (**BinaryDataBuffer** &is)
- virtual void **write** (**BinaryDataBuffer** &os) const

  Contructors

  - **RwyStatusMsg** ()
  - **RwyStatusMsg** (const **RwyStatusList** &rsl)
  - **RwyStatusMsg** (const **RwyStatusMsg** &rhs)

  Accessors

  - const **RwyStatusList** & **getRwyStatuses** () const

Private Attributes

- **RwyStatusList rwy_statuses_**

Detailed Description

This message, sent from the core to clients, conveys the identifiers of configured runways and the WTMD status of each runway. It is sent whenever a status changes or an interval timer expires.

Member Typedef Documentation

typedef RwyStatusMgr::RwyStatusList RwyStatusList

Type for holding the set of `RunwayStatus` objects.

Constructor & Destructor Documentation

RwyStatusMsg ()

Default constructor is intended solely for use by **`Msg::extract()`**.

RwyStatusMsg (const RwyStatusList & *rsl*)

Construct a `RunwayStatusMsg` using the information that it will convey.

RwyStatusMsg (const RwyStatusMsg & *rhs*)

Copy constructor.

~RwyStatusMsg () `[virtual]`

Destructor just implicitly invokes destructors of members and base class.

Member Function Documentation

Msg * duplicate () const `[virtual]`

Return an exact copy of self allocated on heap.

Implements **Msg** (*p.100*).bool read (BinaryDataBuffer & *is*) `[virtual]`

Extract **Msg** contents from a buffer. Derived classes should call parent class **read()** before removing data. Returns `true` iff extraction OK.

Reimplemented from **Msg** (*p.100*).void write (BinaryDataBuffer & *os*) const `[virtual]`

Insert **Msg** contents into a buffer. Derived classes should call parent class **write()** before adding data.

Reimplemented from **Msg** (*p.100*).const RwyStatusList& getRwyStatuses () const

Return the list of runway statuses.

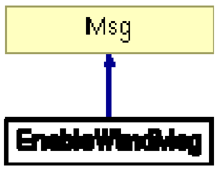
Member Data Documentation

RwyStatusList rwy_statuses_ `[private]`

The list of runway statuses.

6.5.4.3 EnableWtmdMsg

Inheritance diagram for EnableWtmdMsg:



Public Member Functions

- virtual **~EnableWtmdMsg** ()
- virtual **Msg** * **duplicate** () const
- virtual bool **read** (**BinaryDataBuffer** &is)
- virtual void **write** (**BinaryDataBuffer** &os) const

Contructors

- **EnableWtmdMsg** ()
- **EnableWtmdMsg** (const std::string &rwy_id)
- **EnableWtmdMsg** (const **EnableWtmdMsg** &rhs)

Accessors

- const std::string & **getRwyId** () const

Private Attributes

- std::string **rwy_id_**

Detailed Description

This message, sent from clients to the core, conveys the identifier of a runway for which wake independent operations should be enabled. It is sent when a display client receives the appropriate user input.

Constructor & Destructor Documentation

EnableWtmdMsg ()

Default constructor is intended solely for use by `Msg::extract()`.

EnableWtmdMsg (const std::string & *rwy_id*)

Construct an `EnableWtmdMsg` using the information that it will convey.

EnableWtmdMsg (const EnableWtmdMsg & *rhs*)

Copy constructor.

~EnableWtmdMsg () `[virtual]`

Destructor just implicitly invokes destructors of members and base class.

Member Function Documentation

Msg * duplicate () const `[virtual]`

Return an exact copy of self allocated on heap.

Implements **Msg** (*p.100*).bool read (BinaryDataBuffer & *is*) `[virtual]`

Extract **Msg** contents from a buffer. Derived classes should call parent class **read()** before removing data. Returns `true` iff extraction OK.

Reimplemented from **Msg** (*p.100*).void write (BinaryDataBuffer & *os*) const `[virtual]`

Insert **Msg** contents into a buffer. Derived classes should call parent class **write()** before adding data.

Reimplemented from **Msg** (*p.100*).const std::string& getRwyId () const

Return the Id of the runway to enable.

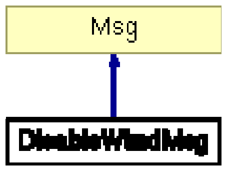
Member Data Documentation

std::string rwy_id_ `[private]`

The Id of the runway to enable.

6.5.4.4 DisableWtmdMsg

Inheritance diagram for DisableWtmdMsg:



Public Member Functions

- virtual **~DisableWtmdMsg** ()
- virtual **Msg** * **duplicate** () const
- virtual bool **read** (**BinaryDataBuffer** &is)
- virtual void **write** (**BinaryDataBuffer** &os) const

  Contructors

  - **DisableWtmdMsg** ()
  - **DisableWtmdMsg** (const std::string &rwy_id)
  - **DisableWtmdMsg** (const **DisableWtmdMsg** &rhs)

  Accessors

  - const std::string & **getRwyId** () const

Private Attributes

- std::string **rwy_id_**

Detailed Description

This message, sent from clients to the core, conveys the identifier of a runway for which wake independent operations should be disabled. It is sent when a display client receives the appropriate user input.

Constructor & Destructor Documentation

DisableWtmdMsg ()

Default constructor is intended solely for use by **`Msg::extract()`**.

DisableWtmdMsg (const std::string & *rwy_id*)

Construct a **`DisableWtmdMsg`** using the information that it will convey.

DisableWtmdMsg (const DisableWtmdMsg & *rhs*)

Copy constructor.

~DisableWtmdMsg () `[virtual]`

Destructor just implicitly invokes destructors of members and base class.

Member Function Documentation

Msg * duplicate () const `[virtual]`

Return an exact copy of self allocated on heap.

Implements **Msg** (*p.100*).bool read (BinaryDataBuffer & *is*) `[virtual]`

Extract **Msg** contents from a buffer. Derived classes should call parent class **read()** before removing data. Returns `true` iff extraction OK.

Reimplemented from **Msg** (*p.100*).void write (BinaryDataBuffer & *os*) const `[virtual]`

Insert **Msg** contents into a buffer. Derived classes should call parent class **write()** before adding data.

Reimplemented from **Msg** (*p.100*).const std::string& getRwyId () const

Return the Id of the runway to disable.

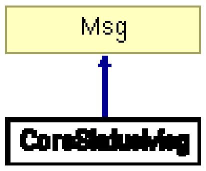
Member Data Documentation

std::string rwy_id_ `[private]`

The Id of the runway to disable.

6.5.4.5 CoreStatusMsg

Inheritance diagram for CoreStatusMsg:



Public Member Functions

- virtual **~CoreStatusMsg** ()
- virtual **Msg** * **duplicate** () const
- virtual bool **read** (**BinaryDataBuffer** &is)
- virtual void **write** (**BinaryDataBuffer** &os) const

  Contructors

  - **CoreStatusMsg** ()
  - **CoreStatusMsg** (**SystemStateMgr::State** status, const **SystemStateMgr::FaultList** &faults)
  - **CoreStatusMsg** (const **CoreStatusMsg** &rhs)

  Accessors

  - **SystemStateMgr::State getStatus** () const
  - const **SystemStateMgr::FaultList** & **getFaultList** () const

Private Attributes

- **SystemStateMgr::State status_**
- **SystemStateMgr::FaultList faults_**

Detailed Description

This message, sent from the core to clients, conveys the overall health of the WTMD system. It contains both the summary status enumeration and the detailed list of active faults. This message also serves as a heartbeat from the core to the clients. It is sent whenever the status changes or an interval timer expires.

Constructor & Destructor Documentation

CoreStatusMsg ()

Default constructor is intended solely for use by `Msg::extract()`.

CoreStatusMsg (SystemStateMgr::State *status*, const SystemStateMgr::FaultList & *faults*)

Construct a `CoreStatusMsg` using the information that it will convey.

CoreStatusMsg (const CoreStatusMsg & *rhs*)

Copy constructor.

~CoreStatusMsg () `[virtual]`

Destructor just implicitly invokes destructors of members and base class.

Member Function Documentation

Msg * duplicate () const `[virtual]`

Return an exact copy of self allocated on heap.

Implements **Msg** (*p.100*).bool read (BinaryDataBuffer & *is*) `[virtual]`

Extract **Msg** contents from a buffer. Derived classes should call parent class **read()** before removing data. Returns `true` iff extraction OK.

Reimplemented from **Msg** (*p.100*).void write (BinaryDataBuffer & *os*) const `[virtual]`

Insert **Msg** contents into a buffer. Derived classes should call parent class **write()** before adding data.

Reimplemented from **Msg** (*p.100*).SystemStateMgr::State getStatus () const

Return the system status.

const SystemStateMgr::FaultList& getFaultList () const

Return the list of system faults.

Member Data Documentation

SystemStateMgr::State status_ `[private]`

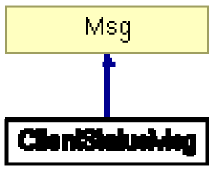
The system status.

SystemStateMgr::FaultList faults_ `[private]`

The list of fault messages.

6.5.4.6 ClientStatusMsg

Inheritance diagram for ClientStatusMsg:



Public Member Functions

- virtual **~ClientStatusMsg** ()
- virtual **Msg** * **duplicate** () const
- virtual bool **read** (**BinaryDataBuffer** &is)
- virtual void **write** (**BinaryDataBuffer** &os) const

Contructors

- **ClientStatusMsg** ()
- **ClientStatusMsg** (**SystemStateMgr::State** status, const std::string &name, const std::string &host, int port, const std::string &failure="")
- **ClientStatusMsg** (const **ClientStatusMsg** &rhs)

Accessors

- **SystemStateMgr::State getStatus** () const
- const std::string & **getName** () const
- const std::string & **getHost** () const
- int **getPort** () const
- const std::string & **getFailure** () const

Private Attributes

- **SystemStateMgr::State status_**
- std::string **name_**
- std::string **host_**
- int **port_**
- std::string **failure_**

Detailed Description

This message, sent from clients to the core, conveys the health of the client. It contains a summary client status enumeration and a text description of an active fault detected by the client (if any). The string **`SystemStateMgr::CORE_TIMEOUT_FAULT`** is used when the client has not received a heartbeat from the core within the timeout interval. This message also contains the name of the client and the host and port number that the core should use to send UDP messages to the client. Finally, this message also serves as a heartbeat from the clients to the core.

Constructor & Destructor Documentation

ClientStatusMsg ()

Default constructor is intended solely for use by `Msg::extract()`.

ClientStatusMsg (SystemStateMgr::State *status*, const std::string & *name*, const std::string & *host*, int *port*, const std::string & *failure* = `""`)

Construct a `ClientStatusMsg` using the information that it will convey.

ClientStatusMsg (const ClientStatusMsg & *rhs*)

Copy constructor.

~ClientStatusMsg () `[virtual]`

Destructor just implicitly invokes destructors of members and base class.

Member Function Documentation

Msg * duplicate () const `[virtual]`

Return an exact copy of self allocated on heap.

Implements **Msg** (*p.100*).bool read (BinaryDataBuffer & *is*) `[virtual]`

Extract **Msg** contents from a buffer. Derived classes should call parent class **read()** before removing data. Returns `true` iff extraction OK.

Reimplemented from **Msg** (*p.100*).void write (BinaryDataBuffer & *os*) const `[virtual]`

Insert **Msg** contents into a buffer. Derived classes should call parent class **write()** before adding data.

Reimplemented from **Msg** (*p.100*).SystemStateMgr::State getStatus () const

Return the client status.

const std::string& getName () const

Return the logical name of client.

const std::string& getHost () const

Return the host name or IP address (#.#.#.#) of client.

int getPort () const

Return the UDP port number client is listenning to.

const std::string& getFailure () const

If status = FAILED, Return the client fault description, else return the empty string.

Member Data Documentation

SystemStateMgr::State status_ `[private]`

The client status.

std::string name_ `[private]`

The logical name of client.

std::string host_ `[private]`

The host name or IP address (#.#.#.#) of client.

int port_ `[private]`

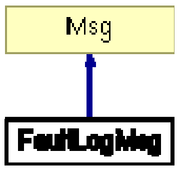
The UDP port number client is listenning to.

std::string failure_ `[private]`

If status = FAILED, the client fault description, else the empty string.

6.5.4.7 FaultLogMsg

Inheritance diagram for FaultLogMsg:



Public Member Functions

- virtual **~FaultLogMsg** ()
- virtual **Msg** * **duplicate** () const
- virtual bool **read** (**BinaryDataBuffer** &is)
- virtual void **write** (**BinaryDataBuffer** &os) const

  Contructors

  - **FaultLogMsg** ()
  - **FaultLogMsg** (const **SystemStateMgr::FaultList** &log)
  - **FaultLogMsg** (const **FaultLogMsg** &rhs)

  Accessors

  - const **SystemStateMgr::FaultList** & **getFaultLog** () const

Private Attributes

- **SystemStateMgr::FaultList fault_log_**

Detailed Description

This message, sent from the core to clients, conveys a complete list of the most recently cleared faults (up to `MAX_LOG_FAULTS`). It is sent whenever a new client checks in so that the new client will have a complete copy of the fault log.

Constructor & Destructor Documentation

FaultLogMsg ()

Default constructor is intended solely for use by **`Msg::extract()`**.

FaultLogMsg (const SystemStateMgr::FaultList & *log*)

Construct a **`FaultLogMsg`** using the information that it will convey.

FaultLogMsg (const FaultLogMsg & *rhs*)

Copy constructor.

~FaultLogMsg () `[virtual]`

Destructor just implicitly invokes destructors of members and base class.

Member Function Documentation

Msg * duplicate () const `[virtual]`

Return an exact copy of self allocated on heap.

Implements **Msg** (*p.100*).bool read (BinaryDataBuffer & *is*) `[virtual]`

Extract **Msg** contents from a buffer. Derived classes should call parent class **read()** before removing data. Returns `true` iff extraction OK.

Reimplemented from **Msg** (*p.100*).void write (BinaryDataBuffer & *os*) const `[virtual]`

Insert **Msg** contents into a buffer. Derived classes should call parent class **write()** before adding data.

Reimplemented from **Msg** (*p.100*).const SystemStateMgr::FaultList& getFaultLog () const

Return the list of fault messages.

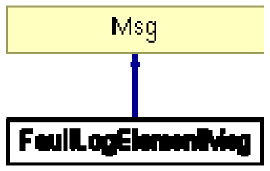
Member Data Documentation

SystemStateMgr::FaultList fault_log_ `[private]`

The list of fault messages.

6.5.4.8 FaultLogElementMsg

Inheritance diagram for FaultLogElementMsg:



Public Member Functions

- **FaultLogElementMsg** ()
- **FaultLogElementMsg** (const **SystemStateMgr::Fault** &flt)
- **FaultLogElementMsg** (const **FaultLogElementMsg** &rhs)
- virtual **~FaultLogElementMsg** ()
- virtual **Msg** * **duplicate** () const
- virtual bool **read** (**BinaryDataBuffer** &is)
- virtual void **write** (**BinaryDataBuffer** &os) const

  Accessors

  - const **SystemStateMgr::Fault** & **getFault** () const

Private Attributes

- **SystemStateMgr::Fault fault_**

Detailed Description

This message, sent from the core to clients, is sent when an active fault has been cleared so that the clients can incrementally maintain their local copy of the fault log.

Constructor & Destructor Documentation

FaultLogElementMsg ()

Default constructor is intended solely for use by **`Msg::extract()`**.

FaultLogElementMsg (const SystemStateMgr::Fault & *flt*)

Construct a **`FaultLogElementMsg`** using the information that it will convey.

FaultLogElementMsg (const FaultLogElementMsg & *rhs*)

Copy constructor.

~FaultLogElementMsg () `[virtual]`

Destructor just implicitly invokes destructors of members and base class.

Member Function Documentation

Msg * duplicate () const `[virtual]`

Return an exact copy of self allocated on heap.

Implements **Msg** (*p.100*).bool read (BinaryDataBuffer & *is*) `[virtual]`

Extract **Msg** contents from a buffer. Derived classes should call parent class **read()** before removing data. Returns `true` iff extraction OK.

Reimplemented from **Msg** (*p.100*).void write (BinaryDataBuffer & *os*) const `[virtual]`

Insert **Msg** contents into a buffer. Derived classes should call parent class **write()** before adding data.

Reimplemented from **Msg** (*p.100*).const SystemStateMgr::Fault& getFault () const

Return the new fault log element.

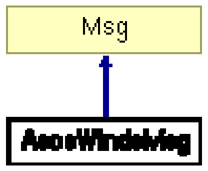
Member Data Documentation

SystemStateMgr::Fault fault_ `[private]`

The new fault log element.

6.5.4.9 AsosWindsMsg

Inheritance diagram for AsosWindsMsg:



Public Member Functions

- virtual **~AsosWindsMsg** ()
- virtual **Msg** * **duplicate** () const
- virtual bool **read** (**BinaryDataBuffer** &is)
- virtual void **write** (**BinaryDataBuffer** &os) const

  Contructors

  - **AsosWindsMsg** ()
  - **AsosWindsMsg** (const **WakeVAS::AbsoluteTime** &time, const **WakeVAS::Speed** &speed, const **WakeVAS::Angle** &direction)
  - **AsosWindsMsg** (const **AsosWindsMsg** &rhs)

  Accessors

  - const **WakeVAS::AbsoluteTime** & **getTime** () const
  - const **WakeVAS::Speed** & **getWindSpeed** () const
  - const **WakeVAS::Angle** & **getWindDirection** () const

Private Attributes

- **WakeVAS::AbsoluteTime time_**
- **WakeVAS::Speed speed_**
- **WakeVAS::Angle direction_**

Detailed Description

This message, sent from the core to clients, conveys ASOS wind information received by the core. It is sent whenever the core processes a new ASOS update.

Constructor & Destructor Documentation

AsosWindsMsg ()

Default constructor is intended solely for use by `Msg::extract()`.

AsosWindsMsg (const WakeVAS::AbsoluteTime & *time*, const WakeVAS::Speed & *speed*, const WakeVAS::Angle & *direction*)

Construct an `AsosWindMsg` using the information that it will convey.

AsosWindsMsg (const AsosWindsMsg & *rhs*)

Copy constructor.

~AsosWindsMsg () `[virtual]`

Destructor just implicitly invokes destructors of members and base class.

Member Function Documentation

Msg * duplicate () const `[virtual]`

Return an exact copy of self allocated on heap.

Implements **Msg** (*p.100*).bool read (BinaryDataBuffer & *is*) `[virtual]`

Extract **Msg** contents from a buffer. Derived classes should call parent class **read()** before removing data. Returns `true` iff extraction OK.

Reimplemented from **Msg** (*p.100*).void write (BinaryDataBuffer & *os*) const `[virtual]`

Insert **Msg** contents into a buffer. Derived classes should call parent class **write()** before adding data.

Reimplemented from **Msg** (*p.100*).const WakeVAS::AbsoluteTime& getTime () const

Return the time of measurement.

const WakeVAS::Speed& getWindSpeed () const

Return the measured wind speed.

const WakeVAS::Angle& getWindDirection () const

Return the measured wind direction.

Member Data Documentation

WakeVAS::AbsoluteTime time_ `[private]`

The time of measurement.

WakeVAS::Speed speed_ `[private]`

The measured wind speed.

WakeVAS::Angle direction_ `[private]`

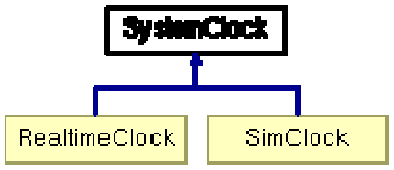
The measured wind direction.

6.5.5 Clocks

These classes provide a generic interface for accessing the “current time” within the prototype. One implementation provides time that is always directly tied to the time held by the computer’s system clock. The second implementation permits a more loosely coupled time source in order to support playback and other non-real-time operations.

6.5.5.1 SystemClock

Inheritance diagram for SystemClock:



Public Member Functions

- virtual **~SystemClock** ()
- virtual **WakeVAS::AbsoluteTime getCurrentTime** () const =0

Static Public Member Functions

- static **SystemClock** & **getInstance** ()

Protected Member Functions

- **SystemClock** ()

Private Member Functions

- **SystemClock** (const **SystemClock** &)
- **SystemClock** & **operator=** (const **SystemClock** &)

Static Private Attributes

- static **SystemClock** * **instance_** = NULL

Detailed Description

This abstract base class is a generic interface to a timekeeping service which can be customized to provide access to the system clock or an alternate time source (e.g. during a scenario playback). This class also implements the singleton pattern.

Constructor & Destructor Documentation

~SystemClock () `[virtual]`

Destructor clears the instance pointer.

SystemClock () `[protected]`

Constructor sets the instance pointer and will assert if another instance already exists. Abstract base class can only be constructed by derived classes.

SystemClock (const SystemClock &) `[private]`

Not allowed and not implemented.

Member Function Documentation

SystemClock & getInstance () `[static]`

Return the singleton instance. Asserts if instance has not been created.

virtual WakeVAS::AbsoluteTime getCurrentTime () const `[pure virtual]`

Concrete implementations of this method will return the current time as defined for their operation.

Implemented in **SimClock** (*p.125*), and **RealtimeClock** (*p.122*).SystemClock& operator= (const SystemClock &) `[private]`

Not allowed and not implemented.

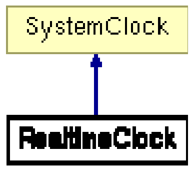
Member Data Documentation

SystemClock * instance_ = NULL `[static, private]`

The singleton instance.

6.5.5.2 RealtimeClock

Inheritance diagram for RealtimeClock:



Public Member Functions

- **RealtimeClock** ()
- virtual **~RealtimeClock** ()
- virtual **WakeVAS::AbsoluteTime getCurrentTime** () const

Private Member Functions

- **RealtimeClock** (const **RealtimeClock** &)
- **RealtimeClock** & **operator=** (const **RealtimeClock** &)

Detailed Description

This derivation of **`SystemClock`** provides a timebase based on the computer's system clock and is thus constrained to operate in real-time only.

Constructor & Destructor Documentation

RealtimeClock ()

Just invokes the base class constructor.

~RealtimeClock () `[virtual]`

Performs no additional work beyond the base class destructor.

RealtimeClock (const RealtimeClock &) `[private]`

Not allowed and not implemented.

Member Function Documentation

WakeVAS::AbsoluteTime getCurrentTime () const `[virtual]`

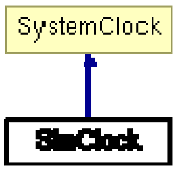
Return an **`AbsoluteTime`** using the current system clock time.

Implements **SystemClock** (*p.121*).RealtimeClock& operator= (const RealtimeClock &) `[private]`

Not allowed and not implemented.

6.5.5.3 SimClock

Inheritance diagram for SimClock:



Public Member Functions

- **SimClock** (double time_factor=1.0)
- virtual **~SimClock** ()
- void **set** (const **AbsoluteTime** &now, double time_factor=1.)
- void **set** (const **AbsoluteTime** &now_wall, const **AbsoluteTime** &now_sim, double time_factor=1.)
- virtual **AbsoluteTime getCurrentTime** () const

Private Member Functions

- **SimClock** (const **SimClock** &)
- **SimClock** & **operator=** (const **SimClock** &)

Private Attributes

- double **time_factor_**
- **AbsoluteTime set_wallclock_**
- **AbsoluteTime set_sim_time_**

Detailed Description

This class is an implementation of a timekeeping service which can be controlled to run at a multiple of real-time from an arbitrary start-time.

Constructor & Destructor Documentation

SimClock (double *time_factor* = `1.0`)

Constructor initializes to run starting at the current syste-clock time. Default time factor of 1.0 will keep synchronized with system clock.

~SimClock () `[virtual]`

Performs no additional work beyond the base class destructor other than implicit member destruction.

SimClock (const SimClock &) `[private]`

Not allowed and not implemented.

Member Function Documentation

void set (const AbsoluteTime & *now*, double *time_factor* = `1.`)

Set simulation clock to `now` effective at the current system clock time and running at a speed of `time_factor`.

void set (const AbsoluteTime & *now_wall*, const AbsoluteTime & *now_sim*, double *time_factor* = `1.`)

Set simulation clock to `now_sim` effective at the system clock time `now_wall` and running at a speed of `time_factor`.

AbsoluteTime getCurrentTime () const `[virtual]`

This implementation returns a time computed according to the formula return = `set_sim_time_` + `time_factor_` * ([current-system-clock-time] - `set_wallclock_`).

Implements **SystemClock** (*p.121*).SimClock& operator= (const SimClock &) `[private]`

Not allowed and not implemented.

Member Data Documentation

double time_factor_ `[private]`

How fast the sim clock runs relative to real-time. 0. stops clock, < 0. would run time backwards!

AbsoluteTime set_wallclock_ `[private]`

System clock time at which simulated time equals `set_sim_time_`.

AbsoluteTime set_sim_time_ `[private]`

The base simulation time.

6.5.6 Socket Wrappers

These classes are used by the implementations of the `ComMgr` to manage the actual socket connections between processes. A buffer class is also provided to facilitate formatting data and moving it to and from the sockets.

6.5.6.1 SocketInterface

Public Types

- typedef int **SocketId**

Public Member Functions

Open/close methods

- **SocketId openSocket** (const **Socket::SocketInfo** &)
- **SocketId openTcpSocket** (const **Socket::SocketInfo** &)
- **SocketId openTcpServerSocket** (const **Socket::SocketInfo** &)
- void **closeSocket** (const **SocketId** socket)
- void **closeAll** ()

Methods for Sending and Receiving

- bool **send** (const **SocketId** socket, const **BinaryDataBuffer** &data)
- bool **getReceivedData** (const **SocketId** socket, **Socket::Queue** &data)

Static Public Member Functions

Singleton pattern methods

- static **SocketInterface** * **instance** ()
- static void **deleteInstance** ()

Static Public Attributes

- static **SocketId INVALID_ID** = -1

Protected Member Functions

- **~SocketInterface** ()

Private Member Functions

- **SocketId addSocket** (**Socket** *socket)
- unsigned int **getNumberOfSockets** ()
- bool **validSocketId** (const **SocketId** &)
- **SocketInterface** (const **SocketInterface** &)
- **SocketInterface** & **operator=** (const **SocketInterface** &)

Private Attributes

- std::vector< **Socket** * > **sockets_**

Static Private Attributes

- static **SocketInterface** * **instance_** = 0

Detailed Description

The **`SocketInterface`** class is a SINGLETON interface providing services to open/close socket connections and send/receive data to/from them.

Member Typedef Documentation

typedef int SocketId

Socket Identifier type.

Constructor & Destructor Documentation

~SocketInterface () `[protected]`

Destructor closes any open Sockets and cleans up.

SocketInterface (const SocketInterface &) `[private]`

Not allowed and not implemented consistent with the singleton pattern.

Member Function Documentation

void deleteInstance () `[static]`

Return pointer to the singleton instance, creating it if necessary. Destroy the singleton instance.

SocketInterface::SocketId openSocket (const Socket::SocketInfo &)

Attempt to open a new UDP socket using the supplied information. Returns a valid SocketId if successful, otherwise `INVALID_ID`.

SocketInterface::SocketId openTcpSocket (const Socket::SocketInfo &)

Attempt to open a new TCP socket using the supplied information. Returns a valid SocketId if successful, otherwise `INVALID_ID`.

SocketInterface::SocketId openTcpServerSocket (const Socket::SocketInfo &)

Open a TCP socket that other applications can connect to. Returns a valid SocketId if successful, otherwise `INVALID_ID`.

void closeSocket (const SocketId *socket*)

close the specified socket (does nothing if socket not valid).

void closeAll ()

Close all Sockets.

bool send (const SocketId *socket*, const BinaryDataBuffer & *data*)

Sends data on the the specified socket. Returns `true` if socket is valid and open for sending

bool getReceivedData (const SocketId *socket*, Socket::Queue & *data*)

Pulls received data from the the specified socket. Returns `true` if socket is valid and open for receiving

SocketInterface::SocketId addSocket (Socket * *socket*) `[private]`

Add socket to the vector of managed sockets and return the SocketId associated with added socket.

unsigned int getNumberOfSockets () `[private]`

Return the number of opened sockets.

bool validSocketId (const SocketId &) `[private]`

Return `true` iff the SocketId is valid.

SocketInterface& operator= (const SocketInterface &) `[private]`

Not allowed and not implemented consistent with the singleton pattern.

Member Data Documentation

SocketInterface::SocketId INVALID_ID = -1 `[static]`

Invalid SocketId value.

std::vector<Socket*> sockets_ `[private]`

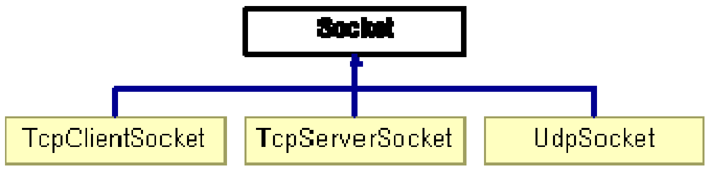
The set of managed **Socket** instances.

SocketInterface * instance_ = 0 `[static, private]`

The singleton instance.

6.5.6.2 Socket

Inheritance diagram for Socket:



Public Classes

- struct **SocketInfo**

Public Types

- typedef enum **WakeVAS::Socket::Direction Direction**
- typedef **WakeVAS::Socket::SocketInfo SocketInfo**
- typedef std::vector< **BinaryDataBuffer** > **Queue**
- enum **Direction** { **INPUT** = 0, **OUTPUT** = 1, **INPUT_OUTPUT** = 2 }

Public Member Functions

- **Socket** (const **SocketInfo** &)
- virtual **~Socket** ()
- virtual bool **initialize** ()=0
- virtual void **send** (const **BinaryDataBuffer** &)=0
- virtual void **getReceivedData** (**Queue** &)=0
- const **Direction** & **getDirection** () const

Protected Member Functions

- std::string **makeFullAddress** (const std::string &ip_adr, const unsigned short &port_num)

Protected Attributes

- **SocketInfo info_**
- ACE_INET_Addr **address_**

Private Member Functions

- **Socket** ()
- **Socket** (const **Socket** &)
- **Socket** & **operator=** (const **Socket** &)

Detailed Description

Socket is a pure virtual base class for a set of simplified wrappers of the ACE socket classes. Derived classes specific to UDP and TCP protocols may start seperate threads for receiving data and listening for connection requests.

Member Typedef Documentation

typedef enum WakeVAS::Socket::Direction Direction

Specification of which direction data will flow through the socket.

typedef struct WakeVAS::Socket::SocketInfo SocketInfo

This structure is used to hold the basic specifications required to open and initialize a socket.

typedef std::vector<BinaryDataBuffer> Queue

Queue is a data structure used to hold one or more chunks of data received by an INPUT or INPUT_OUTPUT **Socket**. Each chunk is stored in a separate **BinaryDataBuffer** and corresponds to the data delivered by a single read() call.

Member Enumeration Documentation

enum Direction

Specification of which direction data will flow through the socket.

Enumerator:

INPUT The socket will only be used for reading data.

OUTPUT The socket will only be used for sending data.

INPUT_OUTPUT The socket will be used for both reading and sending data.

Constructor & Destructor Documentation

Socket (const SocketInfo &)

Constrcutor copies the supplied information and, if the specified `buffer_size` is 0, sets the `buffer_size` to `DEFAULT_BUFFER_SIZE`.

~Socket () `[virtual]`

Base destructor doesn't have anything to do.

Socket () `[private]`

Not allowed and not implemented.

Socket (const Socket &) `[private]`

Not allowed and not implemented.

Member Function Documentation

virtual bool initialize () `[pure virtual]`

Initialize the **Socket** according to the **SocketInfo** provided during construction. This must be implemented by the derived class in order to create the appropriate type of socket. This call may spawn a new thread to handle input or listening for client connections. It returns `true` iff OK.

Implemented in **TcpClientSocket** (*p.140*), **TcpServerSocket** (*p.143*), and **UdpSocket** (*p.137*).virtual void send (const BinaryDataBuffer &) `[pure virtual]`

Send the data in the supplied buffer. Will throw an exception if the underlying socket implementation throws or if not all of the bytes can be sent or the direction was specified as INPUT.

Implemented in **TcpClientSocket** (*p.140*), **TcpServerSocket** (*p.143*), and **UdpSocket** (*p.137*).virtual void getReceivedData (Queue &) `[pure virtual]`

Get the `Queue` of data received since the previous call. Will throw an exception if a fault occurs or the direction was specified as OUTPUT.

const Socket::Direction & getDirection () const

Return the `Direction` specified for this **Socket**.

std::string makeFullAddress (const std::string & *ip_adr*, const unsigned short & *port_num*) `[protected]`

Helper routine produces full ip address format ip_adr:port_num that is expected by the ACE library.

Socket& operator= (const Socket &) `[private]`

Not allowed and not implemented.

Member Data Documentation

SocketInfo info_ `[protected]`

The specification for this **Socket**.

ACE_INET_Addr address_ `[protected]`

The full parsed address as used by the ACE library.

6.5.6.3 Socket::SocketInfo

Public Member Functions

- **SocketInfo** ()
- **~SocketInfo** ()

Public Attributes

- std::string **host_id**
- unsigned short **port_num**
- **Direction dir**
- unsigned int **buffer_size**

Implementation Variables

- static const size_t **DEFAULT_BUFFER_SIZE** = 1024

Detailed Description

This structure is used to hold the basic specifications required to open and initialize a socket.

Constructor & Destructor Documentation

SocketInfo ()

Default constructor.

~SocketInfo ()

Destructor only needs to perform default member destructors.

Member Data Documentation

std::string host_id

Host id: Either an ip address or a name.

unsigned short port_num

Port number.

Direction dir

Direction that data will flow.

unsigned int buffer_size

Maximum receive buffer size.

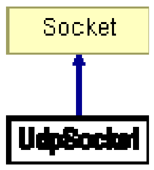
Implementation Variable Documentation

const size_t DEFAULT_BUFFER_SIZE = 1024 `[static]`

The default size for a receive buffer.

6.5.6.4 UdpSocket

Inheritance diagram for UdpSocket:



Public Member Functions

- **UdpSocket** (const **SocketInfo** &)
- virtual **~UdpSocket** ()
- virtual bool **initialize** ()
- virtual void **send** (const **BinaryDataBuffer** &)
- virtual void **getReceivedData** (**Queue** &)

Protected Member Functions

- void **receive** ()

Private Member Functions

- **UdpSocket** ()
- **UdpSocket** (const **UdpSocket** &)
- **UdpSocket** & **operator=** (const **UdpSocket** &)

Static Private Member Functions

- static void * **readThread** (void *arg)

Private Attributes

- char * **recv_buffer_**
- **Queue** **received_data_**
- ACE_Thread_Mutex * **data_mutex_**
- int **read_thread_id_**
- ACE_SOCK_Dgram **endpoint_**
- bool **terminate_**

Detailed Description

The **UdpSocket** class is a simplified wrapper of the ACE `ACE_SOCK_Dgram` class for managing UDP/IP sockets. `UdpSockets` opened for input (or input/output) will spawn a read thread to collect the input data.

Constructor & Destructor Documentation

UdpSocket (const SocketInfo &)

Constrcutor passes the supplied information to the **`Socket`** base class.

~UdpSocket () `[virtual]`

Destructor terminates read thread (if started), closes socket and cleans up buffers.

UdpSocket () `[private]`

Not allowed and not implemented.

UdpSocket (const UdpSocket &) `[private]`

Not allowed and not implemented.

Member Function Documentation

bool initialize () `[virtual]`

Initialize the **Socket** according to the **SocketInfo** provided during construction. This must be implemented by the derived class in order to create the appropriate type of socket. This call may spawn a new thread to handle input or listening for client connections. It returns `true` iff OK.

Implements **Socket** (*p.132*).void send (const BinaryDataBuffer &) `[virtual]`

Send the data in the supplied buffer. Will throw an exception if the underlying socket implementation throws or if not all of the bytes can be sent or the direction was specified as INPUT.

Implements **Socket** (*p.132*).void getReceivedData (Queue &) `[virtual]`

Get the `Queue` of data received since the previous call. Will throw an exception if a fault occurs or the direction was specified as OUTPUT.

void receive () `[protected]`

Method invoked by read thread to pull data from the socket. Returns when data is available or after a 50 mSec timeout interval.

void * readThread (void * *arg*) `[static, private]`

`UdpSocket` read thread method. parameter `arg` is assumed to be a pointer to a **`UdpSocket`**.

UdpSocket& operator= (const UdpSocket &) `[private]`

Not allowed and not implemented.

Member Data Documentation

char* recv_buffer_ `[private]`

Allocated buffer to handle raw input.

Queue received_data_ `[private]`

Queue of received chunks of data stored in `BinaryDataBuffers`.

ACE_Thread_Mutex* data_mutex_ `[private]`

Mutex used to prevent simultaneous access to `received_data_`.

int read_thread_id_ `[private]`

Id of read thread (if started).

ACE_SOCK_Dgram endpoint_ `[private]`

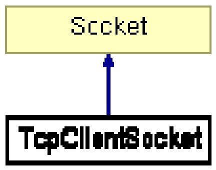
ACE object which manages the socket itself.

bool terminate_ `[private]`

Flag to signal termination to input or connection listening threads.

6.5.6.5 TcpClientSocket

Inheritance diagram for TcpClientSocket:



Public Member Functions

- **TcpClientSocket** (const **SocketInfo** &)
- **TcpClientSocket** (const **SocketInfo** &, ACE_SOCK_Stream *stream)
- **~TcpClientSocket** ()
- virtual bool **initialize** ()
- virtual void **send** (const **BinaryDataBuffer** &)
- virtual void **getReceivedData** (**Queue** &)

Protected Member Functions

- void **receive** ()

Private Member Functions

- **TcpClientSocket** ()
- **TcpClientSocket** (const **TcpClientSocket** &)
- **TcpClientSocket** & **operator=** (const **TcpClientSocket** &)

Static Private Member Functions

- static void * **readThread** (void *arg)

Private Attributes

- char * **recv_buffer_**
- **Queue received_data_**
- ACE_Thread_Mutex * **data_mutex_**
- int **read_thread_id_**
- ACE_SOCK_Stream & **endpoint_**
- bool **terminate_**

Detailed Description

The **TcpClientSocket** class is a simplified wrapper of the ACE `ACE_SOCK_Stream` class for managing TCP/IP sockets. `TcpClientSockets` opened for input (or input/output) will spawn a read thread to collect the input data.

Constructor & Destructor Documentation

TcpClientSocket (const SocketInfo &)

Constrcutor passes the supplied information to the **`Socket`** base class.

TcpClientSocket (const SocketInfo &, ACE_SOCK_Stream * *stream*)

This constructor is for use by **`TcpServerSocket`** when the acceptor returns a new stream. Stream must be a dynamically allocated object which will now be managed by this entity and should already be initialized and connected.

~TcpClientSocket ()

Destructor terminates read thread (if started), closes socket and cleans up buffers.

TcpClientSocket () `[private]`

Not allowed and not implemented.

TcpClientSocket (const TcpClientSocket &) `[private]`

Not allowed and not implemented.

Member Function Documentation

bool initialize () `[virtual]`

Initialize the **Socket** according to the **SocketInfo** provided during construction. This must be implemented by the derived class in order to create the appropriate type of socket. This call may spawn a new thread to handle input or listening for client connections. It returns `true` iff OK.

Implements **Socket** (*p.132*).void send (const BinaryDataBuffer &) `[virtual]`

Send the data in the supplied buffer. Will throw an exception if the underlying socket implementation throws or if not all of the bytes can be sent or the direction was specified as INPUT.

Implements **Socket** (*p.132*).void getReceivedData (Queue &) `[virtual]`

Get the `Queue` of data received since the previous call. Will throw an exception if a fault occurs or the direction was specified as OUTPUT.

void receive () `[protected]`

Method invoked by read thread to pull data from the socket. Returns when data is available or after a 50 mSec timeout interval.

void * readThread (void * *arg*) `[static, private]`

`TcpClientSocket` read thread method. parameter `arg` is assumed to be a pointer to a **`UdpSocket`**.

TcpClientSocket& operator= (const TcpClientSocket &) `[private]`

Not allowed and not implemented.

Member Data Documentation

char* recv_buffer_ `[private]`

Allocated buffer to handle raw input.

Queue received_data_ `[private]`

Queue of received chunks of data stored in `BinaryDataBuffers`.

ACE_Thread_Mutex* data_mutex_ `[private]`

Mutex used to prevent simultaneous access to `received_data_`.

int read_thread_id_ `[private]`

Id of read thread (if started).

ACE_SOCK_Stream& endpoint_ `[private]`

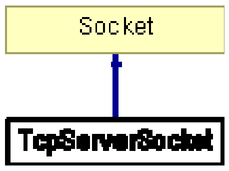
ACE object which manages the socket itself.

bool terminate_ `[private]`

Flag to signal termination to input or connection listening threads.

6.5.6.6 TcpServerSocket

Inheritance diagram for TcpServerSocket:



Public Member Functions

- **TcpServerSocket** (const **SocketInfo** &)
- virtual **~TcpServerSocket** (void)
- virtual bool **initialize** ()
- virtual void **send** (const **BinaryDataBuffer** &)
- virtual void **getReceivedData** (**Queue** &)

Private Types

- typedef std::list< **TcpClientSocket** * > **ConnectionList**
- typedef std::list< ACE_SOCK_Stream * > **NewPeerList**

Private Member Functions

- void **updateConnections** ()
- void **accept** ()
- **TcpServerSocket** (void)
- **TcpServerSocket** (const **TcpServerSocket** &)
- **TcpServerSocket** & **operator=** (const **TcpServerSocket** &)

Static Private Member Functions

- static void * **acceptThread** (void *arg)

Private Attributes

- ACE_SOCK_Acceptor **peer_acceptor_**
- ACE_Thread_Mutex * **accept_mutex_**
- int **accept_thread_id_**
- **NewPeerList new_peers_**
- **ConnectionList connections_**
- bool **terminate_**

Detailed Description

The **TcpServerSocket** class is a simplified wrapper of the ACE `ACE_SOCK_Acceptor` class for managing TCP/IP sockets used for accepting client connections. Upon initialization, this class will spawn an accept thread to wait for connection requests. A **`TcpClientSocket`** is created to manage each connected client. Each connected client will receive all data sent via **`send()`** and all data received is available through the **`getReceivedData()`** call.

Member Typedef Documentation

typedef std::list<TcpClientSocket*> ConnectionList `[private]`

Set for holding connected clients.

typedef std::list<ACE_SOCK_Stream*> NewPeerList `[private]`

Set for holding new streams waiting to be added to connected clients.

Constructor & Destructor Documentation

TcpServerSocket (const SocketInfo &)

Default constructor passes the supplied information to the **`Socket`** base class.

~TcpServerSocket (void) `[virtual]`

Destructor terminates accept thread, closes sockets and cleans up buffers.

TcpServerSocket (void) `[private]`

Not allowed and not implemented.

TcpServerSocket (const TcpServerSocket &) `[private]`

Not allowed and not implemented.

Member Function Documentation

bool initialize () `[virtual]`

Initialize the **Socket** according to the SocketInfo provided during construction. This must be implemented by the derived class in order to create the appropriate type of socket. This call may spawn a new thread to handle input or listening for client connections. It returns `true` iff OK.

Implements **Socket** (*p.132*).void send (const BinaryDataBuffer &) `[virtual]`

Send the data in the supplied buffer to each connected client (after updating the set of connected clients). Will throw an exception if the underlying socket implementation throws or if not all of the bytes can be sent or the direction was specified as INPUT.

Implements **Socket** (*p.132*).void getReceivedData (Queue &) `[virtual]`

Get the `Queue` of data received from all of the connected clients (after updating the set of connected clients) since the previous call. Will throw an exception if a fault occurs or the direction was specified as OUTPUT.

void updateConnections () `[private]`

For each new stream in the `new_peers_` set, create a new **`TcpClientSocket`** to manage that stream and add it to the `connections_` set.

void accept () `[private]`

Method invoked by accept thread to listen for new connections. Returns when a new connection is made or after a 50 mSec timeout interval.

void * acceptThread (void * *arg*) `[static, private]`

`TcpServerSocket` accept thread method. parameter `arg` is assumed to be a pointer to a **`TcpServerSocket`**.

TcpServerSocket& operator= (const TcpServerSocket &) `[private]`

Not allowed and not implemented.

Member Data Documentation

ACE_SOCK_Acceptor peer_acceptor_ `[private]`

ACE object which manages the socket for listening itself.

ACE_Thread_Mutex* accept_mutex_ `[private]`

Mutex used to prevent simultaneous access to `new_peers_`.

int accept_thread_id_ `[private]`

Id of read thread (if started).

NewPeerList new_peers_ `[private]`

The set of new streams waiting to be added to connected clients.

ConnectionList connections_ `[private]`

The set of connected clients.

bool terminate_ `[private]`

Flag to signal termination to connection listening threads.

6.5.6.7 BinaryDataBuffer Class Reference

Public Member Functions

- **~BinaryDataBuffer** ()
- **BinaryDataBuffer** & **operator=** (const **BinaryDataBuffer** &bdb2)
- void **clear** ()

Contructors

- **BinaryDataBuffer** (int size=128)
- **BinaryDataBuffer** (const void *p_data, int num_bytes)
- **BinaryDataBuffer** (const **BinaryDataBuffer** &bdb)

Comparison Methods/operators

- int **contents_are_equal** (**BinaryDataBuffer** &bdb2)
- int **operator==** (**BinaryDataBuffer** &bdb2)
- int **operator!=** (**BinaryDataBuffer** &bdb2)

Accessors

- const void * **p_data** () const
- int **data_size** () const

Methods for Pulling Data from Buffer

- int **extract_data** (void *p_dest, int num_bytes)

Extraction Operators

- **BinaryDataBuffer** & **operator**>> (**BinaryDataBuffer** &s)
- **BinaryDataBuffer** & **operator**>> (char *s)
- **BinaryDataBuffer** & **operator**>> (unsigned char *us)
- **BinaryDataBuffer** & **operator**>> (std::string &s)
- **BinaryDataBuffer** & **operator**>> (unsigned char &uc)
- **BinaryDataBuffer** & **operator**>> (char &c)
- **BinaryDataBuffer** & **operator**>> (short &h)
- **BinaryDataBuffer** & **operator**>> (int &i)
- **BinaryDataBuffer** & **operator**>> (long &l)
- **BinaryDataBuffer** & **operator**>> (unsigned short &uh)
- **BinaryDataBuffer** & **operator**>> (unsigned int &ui)
- **BinaryDataBuffer** & **operator**>> (unsigned long &ul)
- **BinaryDataBuffer** & **operator**>> (float &f)
- **BinaryDataBuffer** & **operator**>> (double &d)

Methods for Adding Data to Buffer

- void **add_data** (const void *p_dest, int num_bytes)

Insertion Operators

- **BinaryDataBuffer** & **operator**<< (const **BinaryDataBuffer** &b)
- **BinaryDataBuffer** & **operator**<< (const char *s)
- **BinaryDataBuffer** & **operator**<< (const unsigned char *us)
- **BinaryDataBuffer** & **operator**<< (const std::string &s)
- **BinaryDataBuffer** & **operator**<< (const unsigned char &uc)
- **BinaryDataBuffer** & **operator**<< (const char &c)
- **BinaryDataBuffer** & **operator**<< (const short &h)
- **BinaryDataBuffer** & **operator**<< (const int &i)
- **BinaryDataBuffer** & **operator**<< (const long &l)

- **BinaryDataBuffer** & **operator<<** (const unsigned short &uh)
- **BinaryDataBuffer** & **operator<<** (const unsigned int &ui)
- **BinaryDataBuffer** & **operator<<** (const unsigned long &ul)
- **BinaryDataBuffer** & **operator<<** (const float &f)
- **BinaryDataBuffer** & **operator<<** (const double &d)

Bulk Write & Read Methods

These methods are complementary pairs for moving data into/out of files and streams or other buffers. Write methods leave the contents of the buffer unchanged. Read methods append the data read to the buffer, growing the internal store if necessary.

- int **write** (int file_desc) const
- int **read** (int file_desc)
- int **write** (int file_desc, int num_bytes) const
- int **read** (int file_desc, int num_bytes)
- int **write** (**BinaryDataBuffer** &bdb) const
- int **read** (**BinaryDataBuffer** &bdb)
- int **write** (**BinaryDataBuffer** &bdb, int num_bytes) const
- int **read** (**BinaryDataBuffer** &bdb, int num_bytes)

Private Member Functions

- void **discard_data** (int bytes_removed)
- void **grow** (int new_size)

Private Attributes

- **BDB_RC_Buff * p_rcbuff_**
- char * **p_insertion_pt**
- char * **p_extraction_pt**
- int **current_data_size**
- int **remaining_buffer_size**

Detailed Description

The **BinaryDataBuffer** class encapsulates a FIFO buffer that defines insertion and extraction methods for adding and extracting intrinsic data types. These methods can be used to isolate applications from byte-order or data-type size inconsistencies when exchanging byte streams between machines of differing architecture (for example, this implementation yields native byte ordering, but the insertion and extraction routines could be easily rewritten (at a small overhead cost) to force either big or little-endian encoding. The **BinaryDataBuffer** also provides efficient buffer copying semantics by performing a shallow copy of a reference-counted data store. The actual data is only copied if an attempt is made to modify it when it is multiply referenced (i.e. copy- on-write semantics). As long as the **BinaryDataBuffer** is not destroyed, the underlying data store is only reallocated when it needs to grow or is written to while multiple references to it are active.

Constructor & Destructor Documentation

BinaryDataBuffer (int *size* = `128`)

Create an empty buffer sized to hold 128 bytes

BinaryDataBuffer (const void * *p_data*, int *num_bytes*)

Construct from supplied data (user formatted).

BinaryDataBuffer (const BinaryDataBuffer & *bdb*)

Copy using copy-on-write symantics.

~BinaryDataBuffer ()

Destructor de-references underlying data store, deallocating it if this is the last reference.

Member Function Documentation

BinaryDataBuffer & operator= (const BinaryDataBuffer & *bdb2*)

Assignment uses copy-on-write symantics.

const void * p_data () const

Get a pointer to the underlying data. Warning! -- this should only be used to perform bulk moves but no attempt should be made to interpret the data unless it was formatted by the user originally.

int data_size () const

Return the number of bytes currently stored.

void clear ()

Empty the buffer of all contents. This does not deallocate or resize the underlying store.

int extract_data (void * *p_dest*, int *num_bytes*)

[in] How many bytes to remove.

Extract bytes from the buffer (effectively removing them). Returns num_bytes or 0 on error (in which case, no bytes are removed). Warning! -- this should only be used to perform bulk moves but no attempt should be made to interpret the data unless it was formatted by the user originally.

Parameters:

p_dest [in] Where to put the data (must be at least num_bytes in size!).

BinaryDataBuffer & operator>> (BinaryDataBuffer & *s*)

Synonymous with write(BinaryDataBuffer& s).

void add_data (const void * *p_dest*, int *num_bytes*)

[in] How many bytes to add.

Insert bytes at the end of the buffer, growing the underlying store if needed. Warning! -- this bypasses the byte ordering and size control of the insertion operators. It is primarily intended for bulk data moves or handling user-formatted data.

Parameters:

p_dest [in] Where to get the data (must be at least num_bytes in size!).

BinaryDataBuffer & operator<< (const BinaryDataBuffer & *b*)

Synonymous with read(const BinaryDataBuffer& s).

int write (int *file_desc*) const

Write to BDB framing controlled stream.

int write (int *file_desc*, int *num_bytes*) const

Write to caller formatted stream.

int write (BinaryDataBuffer & *bdb*) const

Write to BDB framing controlled buffer.

int write (BinaryDataBuffer & *bdb*, int *num_bytes*) const

Write to caller formatted buffer.

void discard_data (int *bytes_removed*) `[private]`

Remove bytes_removed from the front of the buffer. This can be dangerous -- it throws alignment off if the wrong number of bytes is specified.

void grow (int *new_size*) `[private]`

Expand the size of underlying store, copying existing data, if any.

Member Data Documentation

BDB_RC_Buff* p_rcbuff_ `[private]`

Pointer to the underlying store (a reference-counted buffer). Data can only be added if the reference count is 1.

char* p_insertion_pt `[private]`

Where the next inserted byte will be stored.

char* p_extraction_pt `[private]`

Where the next extracted byte will be taken from.

int current_data_size `[private]`

How many bytes are currently available for extraction.

int remaining_buffer_size `[private]`

How many bytes may be inserted before needing to grow the buffer.

6.5.6.8 BDB_RC_Buff Class Reference

Public Member Functions

- **BDB_RC_Buff** (int size)
- char * **p_data** ()
- int **refcount** () const
- void **ref** ()

Static Public Member Functions

- static void **s_unref** (**BDB_RC_Buff** *p_rcb)

Private Member Functions

- **~BDB_RC_Buff** ()
- **BDB_RC_Buff** ()
- **BDB_RC_Buff** (const **BDB_RC_Buff** &)
- **BDB_RC_Buff** & **operator=** (const **BDB_RC_Buff** &)

Private Attributes

- int **refcount_**
- char * **p_storage_**

Detailed Description

This class implements a simple, reference-counted, dynamically-allocated storage buffer for the **BinaryDataBuffer** class. The size of the storage area and where within the storage area data is being inserted or extracted is managed by the **BinaryDataBuffer**.

Constructor & Destructor Documentation

BDB_RC_Buff (int *size*)

Initialize allocating store of indicated size and assuming reference count of 1.

~BDB_RC_Buff () `[private]`

Destructor deletes the allocated store.

BDB_RC_Buff () `[private]`

Not allowed and not implemented.

BDB_RC_Buff (const BDB_RC_Buff &) `[private]`

Not allowed and not implemented.

Member Function Documentation

char* p_data ()

Return pointer to the store.

int refcount () const

Return the reference count.

void ref ()

Note that a new entity is referencing this.

void s_unref (BDB_RC_Buff * *p_rcb*) `[static]`

Note that an entity has de-referenced `p_rcb`, and delete the object if no longer referenced.

BDB_RC_Buff& operator= (const BDB_RC_Buff &) `[private]`

Not allowed and not implemented.

Member Data Documentation

int refcount_ `[private]`

Number of entities referencing this.

char* p_storage_ `[private]`

Pointer to the allocated store.

6.5.7 Physical Quantity Encapsulation

Measurements of angles, speeds, and times within the WTMD prototype are encapsulated by these physical quantity classes. Their use avoids the possibility of measurement unit confusion and permits the definition of certain mathematical operators to permit writing more readable expressions when handling measurements.

6.5.7.1 AbsoluteTime

Public Member Functions

- **AbsoluteTime getMidnightOfSameDay** () const
- **AbsoluteTime getTopOfHour** () const

Contructors

- **AbsoluteTime** (const struct tm &time)
- **AbsoluteTime** (unsigned year, unsigned char month=1, unsigned char day_of_month=1, unsigned char hour=0, unsigned char minute=0, double second=0.)

Comparison methods/operators

- bool **equal** (const **AbsoluteTime** &rhs, const **RelativeTime** &tolerance=**RelativeTime**()) const
- bool **operator<** (const **AbsoluteTime** &rhs) const
- bool **operator<=** (const **AbsoluteTime** &rhs) const
- bool **operator>** (const **AbsoluteTime** &rhs) const
- bool **operator>=** (const **AbsoluteTime** &rhs) const
- bool **operator==** (const **AbsoluteTime** &rhs) const
- bool **operator!=** (const **AbsoluteTime** &rhs) const

Arithmetic operators

- **AbsoluteTime** & **operator+=** (const **RelativeTime** &rhs)
- **AbsoluteTime** & **operator-=** (const **RelativeTime** &rhs)
- **RelativeTime operator-** (const **AbsoluteTime** &rhs) const
- **AbsoluteTime operator-** (const **RelativeTime** &rhs) const
- **AbsoluteTime operator+** (const **RelativeTime** &rhs) const

Accessors

- unsigned **getYear** () const
- unsigned **getMonth** () const
- unsigned **getDayOfMonth** () const
- unsigned **getDayOfYear** () const
- unsigned **getHour** () const
- unsigned **getMinutes** () const
- double **getSeconds** () const

Formatted output methods

- void **printDate** (std::ostream &os) const
- void **printTime** (std::ostream &os) const
- void **print** (std::ostream &os) const
- const char * **toISO8601** (char mid_char= 'T') const

Static Public Member Functions

- static const **AbsoluteTime** & **getJan1_1970_Epoch** ()

- static const **AbsoluteTime getCurrentTime** ()

Protected Member Functions

- **AbsoluteTime** ()
- **AbsoluteTime** (double seconds_since_epoch)

Static Protected Attributes

- static **AbsoluteTime jan_1_1970_epoch_**

Private Member Functions

- void **computeInternalTime** (const struct tm &time)
- void **checkDmyhms** () const
- void **computeDmyhms** () const

Private Attributes

- double **seconds_since_epoch_**
- bool **need_to_compute_dmyhms_**
- double **seconds_**
- unsigned char **minute_**
- unsigned char **hour_**
- unsigned char **day_of_month_**
- unsigned char **month_**
- unsigned **year_**
- unsigned **day_of_year_**

Helper Functions

- AbsoluteTime **WakeVAS::operator+** (const RelativeTime &lhs, const AbsoluteTime &rhs)
- std::ostream & **WakeVAS::operator<<** (std::ostream &os, const AbsoluteTime &t)

Detailed Description

This class represents a specific instance in time which is equivalent to a full specification of year, month, day, hour, minute, and second. This time is assumed to be UTC, not local.

Constructor & Destructor Documentation

AbsoluteTime (const struct tm & *time*)

Construct from a full time specification. See standard documentation of mktime() for fields in struct_tm.

AbsoluteTime (unsigned *year*, unsigned char *month* = `1`, unsigned char *day_of_month* = `1`, unsigned char *hour* = `0`, unsigned char *minute* = `0`, double *second* = `0.`)

Construct from a full time specification.

Parameters:

year >= 1970

month Jan = 1

day_of_month 1-31 (depending on month and year)

hour 0-23

minute 0-59

second 0-59.999...

AbsoluteTime () `[protected]`

Default constructor creates an **AbsoluteTime** equal to the epoch used as the internal reference.

AbsoluteTime (double *seconds_since_epoch*) `[protected]`

Constructor that creates an **AbsoluteTime** based on a time offset relative to the epoch used as the internal reference.

Member Function Documentation

bool equal (const AbsoluteTime & *rhs*, const RelativeTime & *tolerance* = `RelativeTime()`) const

Return `true` iff Times are within tolerance of each other. The tolerance parameter defaults to 0. (i.e. exact equality), but other values can be supplied. A negative tolerance will result in a return value of false.

Parameters:

rhs [in] right-hand-side of comparison

tolerance [in] tolerance to use in comparison (default: 0) bool operator< (const AbsoluteTime & *rhs*) const

Return `true` iff this is less than rhs.

bool operator<= (const AbsoluteTime & *rhs*) const

Return `true` iff this is less than or equal to rhs.

bool operator> (const AbsoluteTime & *rhs*) const

Return `true` iff this is greater than rhs.

bool operator>= (const AbsoluteTime & *rhs*) const

Return `true` iff this is greater than or equal to rhs.

bool operator== (const AbsoluteTime & *rhs*) const

Return `true` iff this is exactly equal to rhs.

bool operator!= (const AbsoluteTime & *rhs*) const

Return `true` iff this is not exactly equal to rhs.

AbsoluteTime & operator+= (const RelativeTime & *rhs*)

Add rhs to this and return this.

AbsoluteTime & operator-= (const RelativeTime & *rhs*)

Subtract rhs from this and return this.

RelativeTime operator- (const AbsoluteTime & *rhs*) const

Subtract rhs from this and return difference

AbsoluteTime operator- (const RelativeTime & *rhs*) const

Subtract rhs from this and return new time

AbsoluteTime operator+ (const RelativeTime & *rhs*) const

Add rhs to this and return the new time

unsigned getYear () const

Returns the full year, e.g. 2003.

unsigned getMonth () const

Returns the month starting with 1 for January.

unsigned getDayOfMonth () const

Jan.= 1.

unsigned getDayOfYear () const

Jan. 1st = 1.

unsigned getHour () const

Get hours since midnight UTC, 0-23.

unsigned getMinutes () const

0-59

double getSeconds () const

0-59.999999999

AbsoluteTime getMidnightOfSameDay () const

Return a time representing midnight of same day as this

AbsoluteTime getTopOfHour () const

Return a time representing current time truncated to the hour -- i.e. set minutes and seconds to 0.

void printDate (std::ostream & *os*) const

Output the date portion in mm/dd/yyyy format.

void printTime (std::ostream & *os*) const

Output the time portion in hh:mm:ss format.

void print (std::ostream & *os*) const

output the time in mm/dd/yyyy hh:mm:ss format.

const char * toISO8601 (char *mid_char* = `'T'`) const

return buffer formatted in ISO 8601 yyyy-mm-ddThh:mm:ss format.

const AbsoluteTime & getJan1_1970_Epoch () `[static]`

Return an **AbsoluteTime** representation of the epoch commonly used in operating systems.

const AbsoluteTime getCurrentTime () `[static]`

Return an **AbsoluteTime** representation of the current system clock.

void computeInternalTime (const struct tm & *time*) `[private]`

Convert a structure specifying all of the parameters of date and time into seconds since the epoch using `mktime()`.

void checkDmyhms () const `[private]`

Recompute the broken-out values if necessary.

void computeDmyhms () const `[private]`

Compute the broken-out values.

Member Data Documentation

AbsoluteTime jan_1_1970_epoch_ `[static, protected]`

Constant representing 1-Jan-1970 (0. seconds since epoch).

double seconds_since_epoch_ `[private]`

This is always kept valid.

bool need_to_compute_dmyhms_ `[mutable, private]`

True iff the day/month/year/etc. values represent the seconds_since_epoch_.

double seconds_ `[mutable, private]`

Seconds portion of **`this`** iff (`need_to_compute_dmyhms_` == `false`) 0-59.9999.

unsigned char minute_ `[mutable, private]`

Minutes portion of **`this`** iff (`need_to_compute_dmyhms_` == `false`) 0-59.

unsigned char hour_ `[mutable, private]`

Hours portion of **`this`** iff (`need_to_compute_dmyhms_` == `false`) 0-23.

unsigned char day_of_month_ `[mutable, private]`

Day-of-month portion of **`this`** iff (`need_to_compute_dmyhms_` == `false`) 1-based.

unsigned char month_ `[mutable, private]`

Month portion of **`this`** iff (`need_to_compute_dmyhms_` == `false`) Jan. = 1.

unsigned year_ `[mutable, private]`

Year portion of **`this`** iff (`need_to_compute_dmyhms_` == `false`) >= 1970.

unsigned day_of_year_ `[mutable, private]`

Day-of-year portion of **`this`** iff (`need_to_compute_dmyhms_` == `false`) Jan. 1st = 1.

Helper Function Documentation

AbsoluteTime **WakeVAS::operator+** (const RelativeTime &lhs, const AbsoluteTime &rhs)

Generate a new **`AbsoluteTime`** by adding a **`RelativeTime`** to **`this`**.

std::ostream & **WakeVAS::operator<<** (std::ostream &os, const AbsoluteTime &t)

Stream insertion operator for printing an **`AbsoluteTime`** to a **`stream`**.

6.5.7.2 Angle

Public Types

- enum **Units** { **RADIANS**, **POSITIVE_RADIANS**, **DEGREES**, **PLUS_MINUS_DEGREES** }

Public Member Functions

Contructors

- **Angle** ()
- **Angle** (double value, **Units** units)

Arithmetic operators

When the result is an ***`Angle`****, it is always normalized to a single turn*

- **Angle** & **operator+=** (const **Angle** &rhs)
- **Angle** & **operator-=** (const **Angle** &rhs)
- **Angle operator-** () const
- **Angle** & **operator *=** (double factor)
- **Angle** & **operator/=** (double factor)
- double **operator/** (const **Angle** &rhs) const

Accessors

- double **getRadians** () const
- double **getPositiveRadians** () const
- double **getDegrees** () const
- unsigned **getWholeDegrees** () const
- unsigned **getTensOfDegrees** () const
- double **getPlusMinusDegrees** () const

String formatting methods

- std::string **getDmsStr** (char neg_label, char pos_label, unsigned seconds_significant_digits=5) const
- std::string **getDmStr** (char neg_label, char pos_label, unsigned minutes_significant_digits=7) const

Trigonometric operations

- double **sine** () const
- double **cosine** () const
- double **tangent** () const

Other manipulations of an angle

- **Angle reciprocal** () const
- **Angle magnitude** () const

Static Public Member Functions

Accessors for commonly used angle constants

- static const **Angle** & **getPi** ()
- static const **Angle** & **getHalfPi** ()

String formatting methods

- static bool **isValidDmsStr** (const std::string &str, char neg_label, char pos_label)
- static **Angle dmsStrToAngle** (const std::string &str, char neg_label, char pos_label)

Static Private Member Functions

- static double **normalize** (double value, **Units** units=RADIANS)

Private Attributes

- double **value_**

Static Private Attributes

- static **Angle s_half_pi_**
- static **Angle s_pi_**

Helper Operations

- const Angle **WakeVAS::operator+** (const Angle &lhs, const Angle &rhs)
- const Angle **WakeVAS::operator-** (const Angle &lhs, const Angle &rhs)
- const Angle **WakeVAS::operator *** (const Angle &lhs, double rhs)
- const Angle **WakeVAS::operator *** (double lhs, const Angle &rhs)
- const Angle **WakeVAS::operator/** (const Angle &lhs, double rhs)

Helper Functions

- **Angle WakeVAS::arcTangent** (double ratio)
- **Angle WakeVAS::arcTangent** (double numerator, double denominator)
- **Angle WakeVAS::arcSine** (double ratio)
- **Angle WakeVAS::arcCosine** (double ratio)

Detailed Description

This class is used to express angular measurments and encapsulates unit conversions and normalization to a single rotation. Because there are no virtual methods, this class should have the same storage overhead as a `double` (except if bloated by RTTI).

Member Enumeration Documentation

enum Units

Enumeration for declaring units used when converting between **`Pressure`** and `double`.

Enumerator:

RADIANS (-PI,PI]

POSITIVE_RADIANS [0,2PI) (technically, non-negative)

DEGREES [0,360)

Constructor & Destructor Documentation

Angle ()

Default constructor creates initializes to a value of 0

Angle (double *value*, Units *units*)

Construct an **Angle** with a given value converted from the specified units to internal representation. Illegal units value will trigger an assert.

Member Function Documentation

Angle & operator+= (const Angle & *rhs*)

Add an angle to this and return this

Angle & operator-= (const Angle & *rhs*)

Subtract an angle from this and return this

Angle operator- () const

Compute the negative of an angle (NOT the reciprocal angle)

Angle & operator *= (double *factor*)

Scale an **Angle** by multiplying it by factor

Angle & operator/= (double *factor*)

Scale an **Angle** by dividing it by factor

double operator/ (const Angle & *rhs*) const

Compute the ratio of two Angles

double getRadians () const

Get angle expressed in range (-PI,PI].

double getPositiveRadians () const

Get angle expressed in range [0,2PI).

double getDegrees () const

Get angle expressed in range [0,360).

unsigned getWholeDegrees () const

Get angle expressed in range [0,359].

unsigned getTensOfDegrees () const

Get angle expressed in range [10,360] rounded to the nearest 10 degrees.

double getPlusMinusDegrees () const

Get angle expressed in range (-180,180].

double sine () const

Return the trigonometric sine of the angle

double cosine () const

Return the trigonometric cosine of the angle

double tangent () const

Return the trigonometric tangent of the angle

Angle reciprocal () const

Compute angle 180 degrees opposite this

Angle magnitude () const

Compute an angle whose value is the absolute value of this angle (e.g. [0,PI] radians or [0,180] degrees

std::string getDmsStr (char *neg_label*, char *pos_label*, unsigned *seconds_significant_digits* = 5) const

Convert **Angle** to a string with degree/minute/decimal-seconds format ([d]dd-mm-ss.ttt{<pos_label>|<neg_label>}) format (e.g. 079-53-22.123W).

Parameters:

neg_label [in] 'S' or 'W'.

pos_label [in] 'N' or 'E'.

seconds_significant_digits [in] valid range is 2-5 (default=5).

std::string getDmStr (char *neg_label*, char *pos_label*, unsigned *minutes_significant_digits* = `7`) const

Convert **Angle** to a string with degree/decimal-minutes format ([d]dd-mm.mmmm{<pos_label>|<neg_label>}) format (e.g. 079-530.36872W).

Parameters:

neg_label [in] 'S' or 'W'.

pos_label [in] 'N' or 'E'.

minutes_significant_digits [in] valid range is 2-7 (default=7).

bool isValidDmsStr (const std::string & *str*, char *neg_label*, char *pos_label*) `[static]`

Verify that a string is in degree/minute/seconds/thousandths ([d]dd-mm-ss[.t[t[t]]]{<pos_label>|<neg_label>}) format (e.g. 079-53-22.123W).

Parameters:

str [in] the string to check formatting on.

neg_label [in] 'S' or 'W'.

pos_label [in] 'N' or 'E'.

Angle dmsStrToAngle (const std::string & *str*, char *neg_label*, char *pos_label*) `[static]`

Convert a string with degree/minute/seconds/thousandths ([d]dd-mm-ss.ttt{<pos_label>|<neg_label>}) format (e.g. 079-53-22.123W) to an **Angle**.

Parameters:

str [in] the string to convert.

neg_label [in] 'S' or 'W'.

pos_label [in] 'N' or 'E'.

const Angle & getPi () `[static]`

Return an **Angle** representing the value PI

const Angle & getHalfPi () `[static]`

Return an **Angle** representing the value PI/2

double normalize (double *value*, Units *units* = `RADIANS`) `[static, private]`

Normalize value to the specified range (+/- or >= 0). Assumes value is already in the proper units (radians or degrees).

Member Data Documentation

double value_ `[private]`

Internally stored in units of RADIANS.

Angle s_half_pi_ `[static, private]`

Angle equal to the constant PI/2.

Angle s_pi_ `[static, private]`

Angle equal to PI.

Helper Operation Documentation

const Angle **WakeVAS::operator+** (const Angle &lhs, const Angle &rhs)

Create an **Angle** that is the sum of two **Angle**s.

const Angle **WakeVAS::operator-** (const Angle &lhs, const Angle &rhs)

Create an **Angle** that is the difference of two **Angle**s.

const Angle **WakeVAS::operator *** (const Angle &lhs, double rhs)

Create an **Angle** that is the product of an **Angle** and a scalar.

const Angle **WakeVAS::operator *** (double lhs, const Angle &rhs)

Create an **Angle** that is the product of an **Angle** and a scalar.

const Angle **WakeVAS::operator/** (const Angle &lhs, double rhs)

Create an **Angle** that is an **Angle** divided by a scalar.

Helper Function Documentation

Angle WakeVAS::arcTangent (double ratio)

Create an **Angle** in [0, PI/2) that is the inverse Tangent of a ratio.

Angle WakeVAS::arcTangent (double numerator, double denominator)

Create an **Angle** in (-PI, PI] that is the inverse Tangent of (**`numerator`**/**`denominator`**) (i.e. four-quadrant inverse tangent).

Angle WakeVAS::arcSine (double ratio)

Create an **Angle** that is the inverse Sine of a ratio.

Angle WakeVAS::arcCosine (double ratio)

Create an **Angle** that is the inverse Cosine of a ratio.

6.5.7.3 RelativeTime

Public Types

- enum **Units** { **SECONDS**, **MINUTES**, **HOURS**, **DAYS**, **MILLISECONDS** }

Public Member Functions

Contructors

- **RelativeTime** ()
- **RelativeTime** (double value, **Units** units)
- **RelativeTime** (const **Distance** &dist, const **Speed** &speed)

Comparison methods/operators

- bool **equal** (const **RelativeTime** &rhs, const **RelativeTime** &tolerance=**RelativeTime**()) const
- bool **operator<** (const **RelativeTime** &rhs) const
- bool **operator<=** (const **RelativeTime** &rhs) const
- bool **operator>** (const **RelativeTime** &rhs) const
- bool **operator>=** (const **RelativeTime** &rhs) const
- bool **operator==** (const **RelativeTime** &rhs) const
- bool **operator!=** (const **RelativeTime** &rhs) const

Arithmetic operators

- **RelativeTime** & **operator+=** (const **RelativeTime** &rhs)
- **RelativeTime** & **operator-=** (const **RelativeTime** &rhs)
- **RelativeTime** & **operator *=** (double scale)
- **RelativeTime** & **operator/=** (double scale)
- **RelativeTime operator-** () const
- double **operator/** (const **RelativeTime** &rhs) const
- **RelativeTime operator+** (const **RelativeTime** &rhs) const
- **RelativeTime operator-** (const **RelativeTime** &rhs) const

Accessors

- double **getValue** (**Units** units) const

Unit-specific Accessors

- double **getMilliSeconds** () const
- double **getSeconds** () const
- double **getMinutes** () const
- double **getHours** () const
- double **getDays** () const

Static Private Member Functions

- static double **getConversionFactor** (**Units** units)
- static double **getInverseConversionFactor** (**Units** units)

Private Attributes

- double **value_**

Helper Operations

- RelativeTime **WakeVAS::operator *** (const RelativeTime &time, double scale)
- RelativeTime **WakeVAS::operator *** (double scale, const RelativeTime &time)
- RelativeTime **WakeVAS::operator+** (const RelativeTime &lhs, const RelativeTime &rhs)
- RelativeTime **WakeVAS::operator-** (const RelativeTime &lhs, const RelativeTime &rhs)
- RelativeTime **WakeVAS::operator/** (const Distance &lhs, const Speed &rhs)
- RelativeTime **WakeVAS::operator/** (const RelativeTime &time, double scale)

Detailed Description

This class represents a time duration or the difference between two absolute times. Because there are no virtual methods, this class should have the same storage overhead as a `double` (except if bloated by RTTI).

Member Enumeration Documentation

enum Units

Enumeration for declaring units used when converting between **`Pressure`** and `double`.

Enumerator:

SECONDS Time duration expressed in seconds.

MINUTES Time duration expressed in minutes.

HOURS Time duration expressed in hours.

DAYS Time duration expressed in days.

Constructor & Destructor Documentation

RelativeTime ()

Default constructor initializes with a value of 0.

RelativeTime (double *value*, Units *units*)

Construct **RelativeTime** converting value to internal format as indicated by units. Illegal units value will trigger an assert.

Parameters:

value [in] initial value

units [in] starting units of initial value

RelativeTime (const Distance & *dist*, const Speed & *speed*)

Construct a **RelativeTime** by dividing distance by speed.

Member Function Documentation

bool equal (const RelativeTime & *rhs*, const RelativeTime & *tolerance* = `RelativeTime()`) const

Return `true` iff RelativeTimes are within tolerance of each other. The tolerance parameter defaults to 0. (i.e. exact equality), but other values can be supplied. A negative tolerance will result in a return value of false.

Parameters:

rhs [in] right-hand-side of comparison

tolerance [in] tolerance to use in comparison (default: 0)

bool operator< (const RelativeTime & *rhs*) const

Return `true` iff this is less than rhs.

bool operator<= (const RelativeTime & *rhs*) const

Return `true` iff this is less than or equal to rhs.

bool operator> (const RelativeTime & *rhs*) const

Return `true` iff this is greater than rhs.

bool operator>= (const RelativeTime & *rhs*) const

Return `true` iff this is greater than or equal to rhs.

bool operator== (const RelativeTime & *rhs*) const

Return `true` iff this is exactly equal to rhs.

bool operator!= (const RelativeTime & *rhs*) const

Return `true` iff this is not exactly equal to rhs.

RelativeTime & operator+= (const RelativeTime & *rhs*)

Add rhs to this and return this.

RelativeTime & operator-= (const RelativeTime & *rhs*)

Subtract rhs from this and return this.

RelativeTime & operator *= (double *scale*)

Multiply this by scale and return this.

RelativeTime & operator/= (double *scale*)

Divide this by scale and return this.

RelativeTime operator- () const

Return a **RelativeTime** with a value equal to the negative of this.

double operator/ (const RelativeTime & *rhs*) const

Compute the ratio of two RelativeTimes.

RelativeTime operator+ (const RelativeTime & *rhs*) const

Compute the sum of two RelativeTimes.

RelativeTime operator- (const RelativeTime & *rhs*) const

Compute the difference of two RelativeTimes.

double getValue (Units *units*) const

Get the value of this converted to specified units. Illegal units value will trigger an assert.

Parameters:

units [in] specification of units of returned value

double getMilliSeconds () const

Get the value of this represented in milliseconds.

double getSeconds () const

Get the value of this represented in seconds.

double getMinutes () const

Get the value of this represented in Minutes.

double getHours () const

Get the value of this represented in hours.

double getDays () const

Get the value of this represented in days.

double getConversionFactor (Units *units*) `[static, private]`

Get the conversion factor from internal units to units. Illegal units specification will trigger an assert.

Parameters:

units [in] units to convert to

double getInverseConversionFactor (Units *units*) `[static, private]`

Get the conversion factor from units to internal units. Illegal units specification will trigger an assert.

Parameters:

units [in] units to convert from

Member Data Documentationdouble value_ `[private]`

Internally stored in units of seconds.

Helper Operations

RelativeTime **WakeVAS::operator *** (const RelativeTime &time, double scale)

Create a **`RelativeTime`** proportional to (scale times) another time duration (ret=time*scale).

RelativeTime **WakeVAS::operator *** (double scale, const RelativeTime &time)

Create a **`RelativeTime`** proportional to (scale times) another time duration (ret=scale*time).

RelativeTime **WakeVAS::operator+** (const RelativeTime &lhs, const RelativeTime &rhs)

Create a **`RelativeTime`** equal to the sum of two **`RelativeTime`**s.

RelativeTime **WakeVAS::operator-** (const RelativeTime &lhs, const RelativeTime &rhs)

Create a **`RelativeTime`** equal to the difference between two **`RelativeTime`**s (ret = lhs – rhs).

RelativeTime **WakeVAS::operator/** (const Distance &lhs, const Speed &rhs)

Create a **`RelativeTime`** by dividing a **`Distance`** by a **`Speed`** (t=d/v).

RelativeTime **WakeVAS::operator/** (const RelativeTime &time, double scale)

Create a **`RelativeTime`** proportional to (inverse of scale times) another time duration (ret=time/scale).

6.5.7.4 Speed

Public Types

- enum **Units** { **KNOTS**, **MPH**, **KM_PER_HOUR**, **FEET_PER_SEC**, **FEET_PER_MIN**, **METERS_PER_SEC** }

Public Member Functions

Contructors

- **Speed** ()
- **Speed** (double value, **Units** units)
- **Speed** (const **Distance** &dist, const **RelativeTime** &time)

Comparison methods/operators

- bool **equal** (const **Speed** &rhs, const **Speed** &tolerance=**Speed**()) const
- bool **operator<** (const **Speed** &rhs) const
- bool **operator<=** (const **Speed** &rhs) const
- bool **operator>** (const **Speed** &rhs) const
- bool **operator>=** (const **Speed** &rhs) const
- bool **operator==** (const **Speed** &rhs) const
- bool **operator!=** (const **Speed** &rhs) const

Arithmetic operators

- **Speed** & **operator+=** (const **Speed** &rhs)
- **Speed** & **operator-=** (const **Speed** &rhs)
- **Speed** & **operator *=** (double scale)
- **Speed** & **operator/=** (double scale)
- **Speed operator-** () const
- **Distance operator *** (const **RelativeTime** &time) const
- double **operator/** (const **Speed** &rhs) const
- **Speed magnitude** () const

Accessors

- double **getValue** (**Units** units) const

Unit-specific Accessors

- double **getKnots** () const
- double **getMPH** () const
- double **getKPH** () const
- double **getFPS** () const
- double **getFPM** () const
- double **getMPS** () const

Static Private Member Functions

- static double **getConversionFactor** (**Units** units)
- static double **getInverseConversionFactor** (**Units** units)

Private Attributes

- double **value_**

Helper Operations

- Speed **WakeVAS::operator/** (const Distance &dist, const RelativeTime &time)
- **Distance WakeVAS::operator *** (const **RelativeTime** &time, const **Speed** &speed)
- Speed **WakeVAS::operator+** (const Speed &lhs, const Speed &rhs)
- Speed **WakeVAS::operator-** (const Speed &lhs, const Speed &rhs)
- Speed **WakeVAS::operator *** (double lhs, const Speed &rhs)
- Speed **WakeVAS::operator *** (const Speed &lhs, double rhs)
- Speed **WakeVAS::operator/** (const Speed &lhs, double rhs)

Detailed Description

This class is used to express speeds and encapsulates unit conversions. Because there are no virtual methods, this class should have the same storage overhead as a `double` (except if bloated by RTTI).

Member Enumeration Documentation

enum Units

Enumeration for declaring units used when converting between **`Speed`** and `double`.

Enumerator:

KNOTS Nautical miles per hour.

MPH Statute miles per hour.

KM_PER_HOUR Kilometers per hour.

FEET_PER_SEC Feet per second.

FEET_PER_MIN Feet per minute.

Constructor & Destructor Documentation

Speed ()

Default constructor creates **Speed** with value of 0.

Speed (double *value*, Units *units*)

Construct **Speed** converting value to internal format as indicated by units. Illegal units value will trigger an assert.

Parameters:

value [in] initial value

units [in] starting units of initial value

Speed (const Distance & *dist*, const RelativeTime & *time*)

Construct speed by dividing distance by delta-time

Member Function Documentation

bool equal (const Speed & *rhs*, const Speed & *tolerance* = `Speed()`) const

Return `true` iff Speeds are within tolerance of each other. The tolerance parameter defaults to 0. (i.e. exact equality), but other values can be supplied. A negative tolerance will result in a return value of false.

Parameters:

rhs [in] right-hand-side of comparison

tolerance [in] tolerance to use in comparison (default: 0)

bool operator< (const Speed & *rhs*) const

Return `true` if this is less than rhs.

bool operator<= (const Speed & *rhs*) const

Return `true` if this is less than or equal to rhs.

bool operator> (const Speed & *rhs*) const

Return `true` if this is greater than rhs.

bool operator>= (const Speed & *rhs*) const

Return `true` if this is greater than or equal to rhs.

bool operator== (const Speed & *rhs*) const

Return `true` if this is exactly equal to rhs.

bool operator!= (const Speed & *rhs*) const

Return `true` if this is not exactly equal to rhs.

Speed & operator+= (const Speed & *rhs*)

Add rhs to this and return this.

Speed & operator-= (const Speed & *rhs*)

Subtract rhs from this and return this.

Speed & operator *= (double *scale*)

Multiply this by scale and return this.

Speed & operator/= (double *scale*)

Divide this by scale and return this.

Speed operator- () const

Return a **Speed** with a value equal to the negative of this.

Distance operator * (const RelativeTime & *time*) const

Return a **Distance** equal to this times delta-time.

double operator/ (const Speed & *rhs*) const

Compute ratio of two Speeds.

Speed magnitude () const

Return a **Speed** with a value equal to the absolute value of this.

double getValue (Units *units*) const

Get the value of this converted to specified units. Illegal units value will trigger an assert.

Parameters:
units [in] specification of units of returned value

double getKnots () const

Get the value of this represented in Natuical Miles per Hour.

double getMPH () const

Get the value of this represented in Statute Miles per Hour.

double getKPH () const

Get the value of this represented in kilometers per Hour.

double getFPS () const

Get the value of this represented in feet per second.

double getFPM () const

Get the value of this represented in feet per minute.

double getMPS () const

Get the value of this represented in meters per second.

double getConversionFactor (Units *units*) `[static, private]`

Get the conversion factor from internal units to units. Illegal units specification will trigger an assert.

Parameters:

units [in] units to convert to

double getInverseConversionFactor (Units *units*) `[static, private]`

Get the conversion factor from units to internal units. Illegal units specification will trigger an assert.

Parameters:

units [in] units to convert from

Member Data Documentationdouble value_ `[private]`

Internally stored in units of Nautical Miles per hour.

Helper Operations

Speed **WakeVAS::operator/** (const Distance &dist, const RelativeTime &time)

Return a **`Speed`** computed by dividing a **`Distance`** by a **`RelativeTime`** (v=d/t).

Distance WakeVAS::operator * (const **RelativeTime** &time, const **Speed** &speed)

Return a **`Distance`** computed by multiplying a **`RelativeTime`** by a **`Speed`** (d=t*v).

Speed **WakeVAS::operator+** (const Speed &lhs, const Speed &rhs)

Return a **`speed`** equal to the sum of two **`Speed`**s (ret=lhs+rhs).

Speed **WakeVAS::operator-** (const Speed &lhs, const Speed &rhs)

Return a **`speed`** equal to the difference of two **`Speed`**s (ret=lhs-rhs).

Speed **WakeVAS::operator *** (double lhs, const Speed &rhs)

Return a **`speed`** proportional to another **`Speed`** (ret=lhs*rhs).

Speed **WakeVAS::operator *** (const Speed &lhs, double rhs)

Return a **`speed`** proportional to another **`Speed`** (ret=lhs*rhs).

Speed **WakeVAS::operator/** (const Speed &lhs, double rhs)

Return a **`speed`** proportional to another **`Speed`** (ret=lhs/rhs).

6.6 Index to Classes, Methods, and Functions

~AsosWindsMsg
  WakeVAS::AsosWindsMsg, 118
~BDB_RC_Buff
  WakeVAS::BDB_RC_Buff, 151
~BinaryDataBuffer
  WakeVAS::BinaryDataBuffer, 148
~ClientComMgr
  WakeVAS::ClientComMgr, 92
~ClientRwyStatusMgr
  WakeVAS::ClientRwyStatusMgr, 69
~ClientStatusMsg
  WakeVAS::ClientStatusMsg, 111
~ClientSystemStateMgr
  WakeVAS::ClientSystemStateMgr, 85
~ComMgr
  WakeVAS::ComMgr, 89
~CoreAppWindow
  CoreAppWindow, 28
~CoreComMgr
  WakeVAS::CoreComMgr, 95
~CoreRwyStatusMgr
  WakeVAS::CoreRwyStatusMgr, 67
~CoreStatusMsg
  WakeVAS::CoreStatusMsg, 109
~CoreSystemStateMgr
  WakeVAS::CoreSystemStateMgr, 80
~DisableWtmdMsg
  WakeVAS::DisableWtmdMsg, 107
~EnableWtmdMsg
  WakeVAS::EnableWtmdMsg, 105
~FaultLogElementMsg
  WakeVAS::FaultLogElementMsg, 116
~FaultLogMsg
  WakeVAS::FaultLogMsg, 114
~LocalAppWindow
  LocalAppWindow, 47
~Msg
  WakeVAS::Msg, 99
~RealtimeClock
  WakeVAS::RealtimeClock, 122
~RwyStatus
  WakeVAS::RwyStatus, 59
~RwyStatusMgr
  WakeVAS::RwyStatusMgr, 63
~RwyStatusMsg
  WakeVAS::RwyStatusMsg, 103
~SimClock
  WakeVAS::SimClock, 124
~Socket
  WakeVAS::Socket, 131
~SocketInfo
  WakeVAS::Socket::SocketInfo, 134
~SocketInterface
  WakeVAS::SocketInterface, 127
~SupervisorAppWindow
  SupervisorAppWindow, 36
~SystemClock
  WakeVAS::SystemClock, 121
~SystemStateMgr
  WakeVAS::SystemStateMgr, 73
~TcpClientSocket
  WakeVAS::TcpClientSocket, 140
~TcpServerSocket
  WakeVAS::TcpServerSocket, 143
~UdpSocket
  WakeVAS::UdpSocket, 137
~WfaWrapper
  WakeVAS::WfaWrapper, 25
~WTMD_CoreApp
  WTMD_CoreApp, 20
~WTMD_IDSGwApp
  WTMD_IDSGwApp, 54
~WTMD_LocalApp
  WTMD_LocalApp, 45
~WTMD_SupervisorApp
  WTMD_SupervisorApp, 34
AbsoluteTime
  WakeVAS::AbsoluteTime, 154, 155
accept
  WakeVAS::TcpServerSocket, 144
accept_mutex_
  WakeVAS::TcpServerSocket, 144
accept_thread_id_
  WakeVAS::TcpServerSocket, 144
acceptThread
  WakeVAS::TcpServerSocket, 144
add_data
  WakeVAS::BinaryDataBuffer, 149
addClient
  WakeVAS::CoreComMgr, 96
addFault
  WakeVAS::ClientSystemStateMgr, 85
  WakeVAS::SystemStateMgr, 74
addLogItem
  WakeVAS::CoreSystemStateMgr, 81
  WakeVAS::SystemStateMgr, 74
address_
  WakeVAS::Socket, 133
addSocket
  WakeVAS::SocketInterface, 128
alert_ostream_
  WTMD_CoreApp, 22

alertDataSet
WfaWrapper.cpp, 27
alertParams
WfaWrapper.cpp, 27
amber
SupervisorDialog.cpp, 42, 51
Angle
WakeVAS::Angle, 162
areAnyAvailable
WakeVAS::RwyStatusMgr, 64
areAnyEnabled
WakeVAS::RwyStatusMgr, 63
asos_direction_
LocalDialog, 50
SupervisorDialog, 40
asos_istream_
WTMD_CoreApp, 22
asos_speed_
LocalDialog, 50
SupervisorDialog, 40
AsosWindsMsg
WakeVAS::AsosWindsMsg, 117, 118
AUDIO_DISABLE_BUTTON_TEXT
SupervisorDialog.cpp, 42
AUDIO_DISABLED_STATUS_TEXT
SupervisorDialog.cpp, 42
AUDIO_ENABLE_BUTTON_TEXT
SupervisorDialog.cpp, 42
AUDIO_ENABLED_STATUS_TEXT
SupervisorDialog.cpp, 42
audio_filename_
SupervisorDialog, 40
audio_system_
SupervisorDialog, 40
available_
WakeVAS::RwyStatus, 60
AVAILABLE_AND_ENABLED_STATUS_TE
XT
SupervisorDialog.cpp, 41, 51
available_duration_total_
WTMD_CoreApp, 22
AVAILABLE_STATUS_TEXT
SupervisorDialog.cpp, 41
available_time_
WTMD_CoreApp, 22
BDB_RC_Buff
WakeVAS::BDB_RC_Buff, 151
BinaryDataBuffer
WakeVAS::BinaryDataBuffer, 148
BLANK_BUTTON_TEXT
SupervisorDialog.cpp, 41, 50
BLANK_STATUS_TEXT
SupervisorDialog.cpp, 42, 51
buffer_size
WakeVAS::Socket::SocketInfo, 135
checkDmyhms
WakeVAS::AbsoluteTime, 158
clear
WakeVAS::BinaryDataBuffer, 148
clearMsgs
WakeVAS::ClientComMgr, 93
WakeVAS::CoreComMgr, 96
CLIENT_HEARTBEAT_TIMEOUT
CoreSystemStateMgr.cpp, 82
CLIENT_REMOVE_TIMEOUT
CoreSystemStateMgr.cpp, 82
ClientComMgr
WakeVAS::ClientComMgr, 92
ClientData
WakeVAS::CoreSystemStateMgr::ClientData, 83
ClientRwyStatusMgr
WakeVAS::ClientRwyStatusMgr, 69
clients_
WakeVAS::CoreComMgr, 97
WakeVAS::CoreSystemStateMgr, 81
ClientSet
WakeVAS::CoreComMgr, 95
WakeVAS::CoreSystemStateMgr, 80
ClientStatusMsg
WakeVAS::ClientStatusMsg, 111
ClientSystemStateMgr
WakeVAS::ClientSystemStateMgr, 85
ClientSystemStateMgr.cpp
CORE_HEARTBEAT_TIMEOUT, 87
HEARTBEAT_INTERVAL, 87
clock_
WTMD_CoreApp, 22
WTMD_IDSGwApp, 55
WTMD_LocalApp, 46
WTMD_SupervisorApp, 35
closeAll
WakeVAS::SocketInterface, 128
closeButtonPressed
CoreAppWindow, 28
LocalAppWindow, 47
SupervisorAppWindow, 36
closeoutReport
WTMD_CoreApp, 21
closeSocket
WakeVAS::SocketInterface, 128
com_mgr_
WTMD_CoreApp, 22
WTMD_IDSGwApp, 55
WTMD_LocalApp, 46
WTMD_SupervisorApp, 35
ComMgr
WakeVAS::ComMgr, 89
components/src/ClientSystemStateMgr.cpp, 84
components/src/CoreRwyStatusMgr.cpp, 66
components/src/CoreSystemStateMgr.cpp, 79
computeDmyhms

WakeVAS::AbsoluteTime, 158
computeInternalTime
WakeVAS::AbsoluteTime, 158
ConnectionList
WakeVAS::TcpServerSocket, 143
connections_
WakeVAS::TcpServerSocket, 144
COORDINATION_PORT_NUM
SupervisorDialog.cpp, 41
core/src/CoreDialog.cpp, 30
CORE_HEARTBEAT_TIMEOUT
ClientSystemStateMgr.cpp, 87
CORE_TIMEOUT_FAULT
WakeVAS::SystemStateMgr, 75
CoreAppWindow, 28
~CoreAppWindow, 28
closeButtonPressed, 28
CoreAppWindow, 28
CoreComMgr
WakeVAS::CoreComMgr, 95
CoreDialog, 30
getAsosDirection, 31
getAsosSpeed, 31
handleQuitButton, 31
handleViewLogButton, 31
setAsosDirection, 31
setAsosSpeed, 31
updateView, 31
CoreDialog.cpp
FAILED_SYSTEM_STATE_TEXT, 31
INITIALIZING_SYSTEM_STATE_TEXT, 31
OPERATIONAL_SYSTEM_STATE_TEXT, 32
CoreRwyStatusMgr
WakeVAS::CoreRwyStatusMgr, 67
CoreRwyStatusMgr.cpp
TX_INTERVAL, 68
CoreStatusMsg
WakeVAS::CoreStatusMsg, 108, 109
CoreSystemStateMgr
WakeVAS::CoreSystemStateMgr, 80
CoreSystemStateMgr.cpp
CLIENT_HEARTBEAT_TIMEOUT, 82
CLIENT_REMOVE_TIMEOUT, 82
HEARTBEAT_INTERVAL, 82
makeUniqueName, 81
cosine
WakeVAS::Angle, 163
current_data_size
WakeVAS::BinaryDataBuffer, 150
currProf
WfaWrapper.cpp, 27
currWind
WfaWrapper.cpp, 27
data_mutex_
WakeVAS::TcpClientSocket, 141
WakeVAS::UdpSocket, 138
data_size
WakeVAS::BinaryDataBuffer, 148
day_of_month_
WakeVAS::AbsoluteTime, 159
day_of_year_
WakeVAS::AbsoluteTime, 159
DAYS
WakeVAS::RelativeTime, 168
DEFAULT_BUFFER_SIZE
Socket.cpp, 135
DEGREES
WakeVAS::Angle, 161
deleteClient
WakeVAS::CoreComMgr, 96
deleteInstance
WakeVAS::SocketInterface, 127
dir
WakeVAS::Socket::SocketInfo, 134
Direction
WakeVAS::Socket, 131
direction_
WakeVAS::AsosWindsMsg, 119
DISABLE_BUTTON_TEXT
SupervisorDialog.cpp, 41
disableAudioSystem
SupervisorDialog, 40
DisableWtmdMsg
WakeVAS::DisableWtmdMsg, 106
discard_data
WakeVAS::BinaryDataBuffer, 149
dmsStrToAngle
WakeVAS::Angle, 164
duplicate
WakeVAS::AsosWindsMsg, 118
WakeVAS::ClientStatusMsg, 111
WakeVAS::CoreStatusMsg, 109
WakeVAS::DisableWtmdMsg, 107
WakeVAS::EnableWtmdMsg, 105
WakeVAS::FaultLogElementMsg, 116
WakeVAS::FaultLogMsg, 114
WakeVAS::Msg, 100
WakeVAS::RwyStatusMsg, 103
ENABLE_BUTTON_TEXT
SupervisorDialog.cpp, 41
enableAudioSystem
SupervisorDialog, 40
enabled_
WakeVAS::RwyStatus, 60
ENABLED_ONLY_STATUS_TEXT
SupervisorDialog.cpp, 41, 51
EnableWtmdMsg
WakeVAS::EnableWtmdMsg, 104
end_time_
WakeVAS::SystemStateMgr::Fault, 77

endpoint_
WakeVAS::TcpClientSocket, 141
WakeVAS::UdpSocket, 138
equal
WakeVAS::AbsoluteTime, 155
WakeVAS::RelativeTime, 169
WakeVAS::Speed, 175
extract
WakeVAS::Msg, 100
extract_data
WakeVAS::BinaryDataBuffer, 148
FAILED
WakeVAS::SystemStateMgr, 73
FAILED_SYSTEM_STATE_TEXT
CoreDialog.cpp, 31
SupervisorDialog.cpp, 42, 51
failure_
WakeVAS::ClientStatusMsg, 112
Fault
WakeVAS::SystemStateMgr::Fault, 76, 77
fault_
WakeVAS::CoreSystemStateMgr::ClientData, 83
WakeVAS::FaultLogElementMsg, 116
fault_log_
WakeVAS::FaultLogMsg, 114
WakeVAS::SystemStateMgr, 75
FaultList
WakeVAS::SystemStateMgr, 72
FaultLogElementMsg
WakeVAS::FaultLogElementMsg, 115
FaultLogMsg
WakeVAS::FaultLogMsg, 113
faults_
WakeVAS::CoreStatusMsg, 109
WakeVAS::SystemStateMgr, 75
FEET_PER_MIN
WakeVAS::Speed, 174
FEET_PER_SEC
WakeVAS::Speed, 174
first_joined_
WakeVAS::CoreSystemStateMgr::ClientData, 83
flushOutput
WakeVAS::ClientComMgr, 92
WakeVAS::ComMgr, 90
WakeVAS::CoreComMgr, 96
gatherInput
WakeVAS::ClientComMgr, 92
WakeVAS::ComMgr, 90
WakeVAS::CoreComMgr, 96
getApplicationName
WTMD_CoreApp, 21
WTMD_IDSGwApp, 54
WTMD_LocalApp, 45
WTMD_SupervisorApp, 34
getApplicationVersion
WTMD_CoreApp, 21
WTMD_IDSGwApp, 54
WTMD_LocalApp, 45
WTMD_SupervisorApp, 34
getAsosDirection
CoreDialog, 31
getAsosSpeed
CoreDialog, 31
getConversionFactor
WakeVAS::RelativeTime, 171
WakeVAS::Speed, 177
getCurrentTime
WakeVAS::AbsoluteTime, 158
WakeVAS::RealtimeClock, 122
WakeVAS::SimClock, 125
WakeVAS::SystemClock, 121
getDayOfMonth
WakeVAS::AbsoluteTime, 157
getDayOfYear
WakeVAS::AbsoluteTime, 157
getDays
WakeVAS::RelativeTime, 171
getDegrees
WakeVAS::Angle, 162
getDirection
WakeVAS::Socket, 132
getDmsStr
WakeVAS::Angle, 163
getDmStr
WakeVAS::Angle, 164
getEndTime
WakeVAS::SystemStateMgr::Fault, 77
getFailure
WakeVAS::ClientStatusMsg, 112
getFault
WakeVAS::FaultLogElementMsg, 116
getFaultList
WakeVAS::CoreStatusMsg, 109
WakeVAS::SystemStateMgr, 73
getFaultLog
WakeVAS::FaultLogMsg, 114
getFPM
WakeVAS::Speed, 177
getFPS
WakeVAS::Speed, 176
getHalfPi
WakeVAS::Angle, 164
getHost
WakeVAS::ClientStatusMsg, 111
getHour
WakeVAS::AbsoluteTime, 157
getHours
WakeVAS::RelativeTime, 170
getInstance
WakeVAS::ComMgr, 89

WakeVAS::CoreComMgr, 95
WakeVAS::CoreRwyStatusMgr, 67
WakeVAS::RwyStatusMgr, 63
WakeVAS::SystemClock, 121
WakeVAS::SystemStateMgr, 73
WakeVAS::WfaWrapper, 25
getInverseConversionFactor
WakeVAS::RelativeTime, 171
WakeVAS::Speed, 177
getJan1_1970_Epoch
WakeVAS::AbsoluteTime, 158
getKnots
WakeVAS::Speed, 176
getKPH
WakeVAS::Speed, 176
getMidnightOfSameDay
WakeVAS::AbsoluteTime, 157
getMilliSeconds
WakeVAS::RelativeTime, 170
getMinutes
WakeVAS::AbsoluteTime, 157
WakeVAS::RelativeTime, 170
getMonth
WakeVAS::AbsoluteTime, 157
getMPH
WakeVAS::Speed, 176
getMPS
WakeVAS::Speed, 177
getName
WakeVAS::ClientStatusMsg, 111
getNumberOfSockets
WakeVAS::SocketInterface, 128
getNumRwys
WakeVAS::RwyStatusMgr, 63
getPi
WakeVAS::Angle, 164
getPlusMinusDegrees
WakeVAS::Angle, 163
getPort
WakeVAS::ClientStatusMsg, 112
getPositiveRadians
WakeVAS::Angle, 162
getRadians
WakeVAS::Angle, 162
getReceivedData
WakeVAS::ClientComMgr, 92
WakeVAS::ComMgr, 89
WakeVAS::CoreComMgr, 95
WakeVAS::Socket, 132
WakeVAS::SocketInterface, 128
WakeVAS::TcpClientSocket, 140
WakeVAS::TcpServerSocket, 144
WakeVAS::UdpSocket, 137
getRwyId
WakeVAS::DisableWtmdMsg, 107
WakeVAS::EnableWtmdMsg, 105
WakeVAS::RwyStatus, 60
getRwyIndex
WakeVAS::RwyStatusMgr, 64
getRwyStatus
WakeVAS::RwyStatusMgr, 63, 64
getRwyStatuses
WakeVAS::RwyStatusMsg, 103
getSeconds
WakeVAS::AbsoluteTime, 157
WakeVAS::RelativeTime, 170
getStartTime
WakeVAS::SystemStateMgr::Fault, 77
getState
WakeVAS::SystemStateMgr, 73
getStatus
WakeVAS::ClientStatusMsg, 111
WakeVAS::CoreStatusMsg, 109
getTensOfDegrees
WakeVAS::Angle, 163
getTime
WakeVAS::AsosWindsMsg, 118
getTopOfHour
WakeVAS::AbsoluteTime, 157
getType
WakeVAS::Msg, 100
getValue
WakeVAS::RelativeTime, 170
WakeVAS::Speed, 176
getWholeDegrees
WakeVAS::Angle, 163
getWindDirection
WakeVAS::AsosWindsMsg, 118
getWindSpeed
WakeVAS::AsosWindsMsg, 118
getYear
WakeVAS::AbsoluteTime, 156
grow
WakeVAS::BinaryDataBuffer, 149
GwUpdateTimer, 57
GwUpdateTimer, 57
parent_, 57
timerCallback, 57
handle
WTMD_CoreApp, 20
handleAsosButton
SupervisorDialog, 39
handleAudioButton
SupervisorDialog, 39
handleQuitButton
CoreDialog, 31
handleRwyButton
LocalDialog, 50
SupervisorDialog, 39
handleStateDetailButton
SupervisorDialog, 39
handleViewLogButton

CoreDialog, 31
SupervisorDialog, 40
HEARTBEAT_INTERVAL
ClientSystemStateMgr.cpp, 87
CoreSystemStateMgr.cpp, 82
host_
WakeVAS::ClientStatusMsg, 112
WakeVAS::ClientSystemStateMgr, 86
host_id
WakeVAS::Socket::SocketInfo, 134
hour_
WakeVAS::AbsoluteTime, 158
HOURS
WakeVAS::RelativeTime, 168
id_
WakeVAS::RwyStatus, 60
info_
WakeVAS::Socket, 133
init_time_
WakeVAS::WfaWrapper, 26
initialise
WTMD_CoreApp, 21
WTMD_IDSGwApp, 54
WTMD_LocalApp, 45
WTMD_SupervisorApp, 34
initialize
WakeVAS::ClientComMgr, 92
WakeVAS::CoreComMgr, 95
WakeVAS::CoreRwyStatusMgr, 67
WakeVAS::Socket, 132
WakeVAS::TcpClientSocket, 140
WakeVAS::TcpServerSocket, 143
WakeVAS::UdpSocket, 137
WakeVAS::WfaWrapper, 25
INITIALIZING
WakeVAS::SystemStateMgr, 73
INITIALIZING_SYSTEM_STATE_TEXT
CoreDialog.cpp, 31
SupervisorDialog.cpp, 42, 51
INPUT
WakeVAS::Socket, 131
INPUT_OUTPUT
WakeVAS::Socket, 131
instance
WTMD_CoreApp, 20
instance_
WakeVAS::ComMgr, 90
WakeVAS::CoreComMgr, 97
WakeVAS::CoreRwyStatusMgr, 68
WakeVAS::RwyStatusMgr, 65
WakeVAS::SocketInterface, 129
WakeVAS::SystemClock, 121
WakeVAS::SystemStateMgr, 75
WakeVAS::WfaWrapper, 26
WTMD_CoreApp, 23
INVALID_ID
WakeVAS::SocketInterface, 128
isAvailable
WakeVAS::RwyStatus, 60
isEnabled
WakeVAS::RwyStatus, 60
isValidDmsStr
WakeVAS::Angle, 164
jan_1_1970_epoch_
WakeVAS::AbsoluteTime, 158
KM_PER_HOUR
WakeVAS::Speed, 174
KNOTS
WakeVAS::Speed, 174
last_heartbeat_rx_
WakeVAS::ClientSystemStateMgr, 86
last_heartbeat_tx_
WakeVAS::ClientSystemStateMgr, 86
WakeVAS::CoreSystemStateMgr, 81
last_msg_tx_
WakeVAS::CoreRwyStatusMgr, 68
last_update_
WakeVAS::CoreSystemStateMgr::ClientData, 83
local_faults_
WakeVAS::ClientSystemStateMgr, 86
LocalAppWindow, 47
~LocalAppWindow, 47
closeButtonPressed, 47
LocalAppWindow, 47
LocalDialog, 48
asos_direction_, 50
asos_speed_, 50
handleRwyButton, 50
rwy_buttons, 50
rwy_ids, 50
rwy_statuses, 50
update, 49
updater_, 50
updateView, 49
LocalDialog::UpdateTimer, 52
parent_, 52
timerCallback, 52
UpdateTimer, 52
magnitude
WakeVAS::Angle, 163
WakeVAS::Speed, 176
makeFullAddress
WakeVAS::Socket, 132
makeUniqueName
CoreSystemStateMgr.cpp, 81
MAX_ASOS_AGE
WfaWrapper.cpp, 27
MAX_ASOS_AGE_SECS
WfaWrapper.cpp, 26
MAX_LOG_FAULTS
SystemStateMgr.cpp, 75

MAX_RUC_AGE
  WfaWrapper.cpp, 27
minute_
  WakeVAS::AbsoluteTime, 158
MINUTES
  WakeVAS::RelativeTime, 168
month_
  WakeVAS::AbsoluteTime, 159
moreThanOneInstanceAllowed
  WTMD_CoreApp, 21
  WTMD_IDSGwApp, 54
  WTMD_LocalApp, 45
  WTMD_SupervisorApp, 34
MPH
  WakeVAS::Speed, 174
Msg
  WakeVAS::Msg, 100
msg_type_
  WakeVAS::Msg, 101
MsgList
  WakeVAS::ComMgr, 89
name_
  WakeVAS::ClientStatusMsg, 112
  WakeVAS::ClientSystemStateMgr, 86
  WTMD_LocalApp, 46
  WTMD_SupervisorApp, 35
need_to_compute_dmyhms_
  WakeVAS::AbsoluteTime, 158
need_to_send_
  WakeVAS::CoreRwyStatusMgr, 68
new_peers_
  WakeVAS::TcpServerSocket, 144
NewPeerList
  WakeVAS::TcpServerSocket, 143
normalize
  WakeVAS::Angle, 165
noteEndTime
  WakeVAS::SystemStateMgr::Fault, 77
offline_
  WTMD_CoreApp, 21
openSocket
  WakeVAS::SocketInterface, 127
openTcpServerSocket
  WakeVAS::SocketInterface, 128
openTcpSocket
  WakeVAS::SocketInterface, 127
OPERATIONAL
  WakeVAS::SystemStateMgr, 73
operational_duration_total_
  WTMD_CoreApp, 22
OPERATIONAL_SYSTEM_STATE_TEXT
  CoreDialog.cpp, 32
  SupervisorDialog.cpp, 42, 51
operational_time_
  WTMD_CoreApp, 22
operator-
  WakeVAS::AbsoluteTime, 156
  WakeVAS::Angle, 162
  WakeVAS::RelativeTime, 170
  WakeVAS::Speed, 176
operator *
  WakeVAS::Speed, 176
operator *=
  WakeVAS::Angle, 162
  WakeVAS::RelativeTime, 170
  WakeVAS::Speed, 176
operator!=
  WakeVAS::AbsoluteTime, 156
  WakeVAS::RelativeTime, 169
  WakeVAS::RwyStatus, 60
  WakeVAS::Speed, 175
operator/
  WakeVAS::Angle, 162
  WakeVAS::RelativeTime, 170
  WakeVAS::Speed, 176
operator/=
  WakeVAS::Angle, 162
  WakeVAS::RelativeTime, 170
  WakeVAS::Speed, 176
operator+
  WakeVAS::AbsoluteTime, 156
  WakeVAS::RelativeTime, 170
operator+=
  WakeVAS::AbsoluteTime, 156
  WakeVAS::Angle, 162
  WakeVAS::RelativeTime, 169
  WakeVAS::Speed, 175
operator<
  WakeVAS::AbsoluteTime, 156
  WakeVAS::RelativeTime, 169
  WakeVAS::Speed, 175
operator<<
  WakeVAS::BinaryDataBuffer, 149
operator<=
  WakeVAS::AbsoluteTime, 156
  WakeVAS::RelativeTime, 169
  WakeVAS::Speed, 175
operator=
  WakeVAS::BDB_RC_Buff, 152
  WakeVAS::BinaryDataBuffer, 148
  WakeVAS::ClientComMgr, 93
  WakeVAS::ClientRwyStatusMgr, 70
  WakeVAS::ClientSystemStateMgr, 86
  WakeVAS::ComMgr, 90
  WakeVAS::CoreComMgr, 96
  WakeVAS::CoreRwyStatusMgr, 68
  WakeVAS::CoreSystemStateMgr, 81
  WakeVAS::RealtimeClock, 123
  WakeVAS::RwyStatus, 59
  WakeVAS::RwyStatusMgr, 64
  WakeVAS::SimClock, 125
  WakeVAS::Socket, 132

WakeVAS::SocketInterface, 128
WakeVAS::SystemClock, 121
WakeVAS::SystemStateMgr, 74
WakeVAS::TcpClientSocket, 141
WakeVAS::TcpServerSocket, 144
WakeVAS::UdpSocket, 138
WakeVAS::WfaWrapper, 26
WTMD_CoreApp, 21
WTMD_IDSGwApp, 55
WTMD_LocalApp, 46
WTMD_SupervisorApp, 35
operator-=
WakeVAS::AbsoluteTime, 156
operator-=
WakeVAS::Angle, 162
operator-=
WakeVAS::RelativeTime, 170
operator-=
WakeVAS::Speed, 176
operator==
WakeVAS::AbsoluteTime, 156
WakeVAS::RelativeTime, 169
WakeVAS::RwyStatus, 59
WakeVAS::Speed, 175
operator>
WakeVAS::AbsoluteTime, 156
WakeVAS::RelativeTime, 169
WakeVAS::Speed, 175
operator>=
WakeVAS::AbsoluteTime, 156
WakeVAS::RelativeTime, 169
WakeVAS::Speed, 175
operator>>
WakeVAS::BinaryDataBuffer, 149
OUTPUT
WakeVAS::Socket, 131
p_data
WakeVAS::BDB_RC_Buff, 152
WakeVAS::BinaryDataBuffer, 148
p_extraction_pt
WakeVAS::BinaryDataBuffer, 150
p_insertion_pt
WakeVAS::BinaryDataBuffer, 150
p_rcbuff_
WakeVAS::BinaryDataBuffer, 150
p_storage_
WakeVAS::BDB_RC_Buff, 152
parent_
GwUpdateTimer, 57
LocalDialog::UpdateTimer, 52
SupervisorDialog::UpdateTimer, 43
UpdateTimer, 29
peer_acceptor_
WakeVAS::TcpServerSocket, 144
port_
WakeVAS::ClientStatusMsg, 112
port_num
WakeVAS::Socket::SocketInfo, 134
POSITIVE_RADIANS
WakeVAS::Angle, 161
print
WakeVAS::AbsoluteTime, 157
WakeVAS::SystemStateMgr::Fault, 77
printDate
WakeVAS::AbsoluteTime, 157
printTime
WakeVAS::AbsoluteTime, 157
profile_istream_
WTMD_CoreApp, 22
putData
WakeVAS::SystemStateMgr, 74
putLogData
WakeVAS::SystemStateMgr, 74
putState
WakeVAS::ClientSystemStateMgr, 85
WakeVAS::CoreSystemStateMgr, 80
WakeVAS::SystemStateMgr, 74
Queue
WakeVAS::Socket, 131
RADIANS
WakeVAS::Angle, 161
read
WakeVAS::AsosWindsMsg, 118
WakeVAS::ClientStatusMsg, 111
WakeVAS::CoreStatusMsg, 109
WakeVAS::DisableWtmdMsg, 107
WakeVAS::EnableWtmdMsg, 105
WakeVAS::FaultLogElementMsg, 116
WakeVAS::FaultLogMsg, 114
WakeVAS::Msg, 100
WakeVAS::RwyStatus, 60
WakeVAS::RwyStatusMsg, 103
WakeVAS::SystemStateMgr::Fault, 77
read_thread_id_
WakeVAS::TcpClientSocket, 141
WakeVAS::UdpSocket, 138
readThread
WakeVAS::TcpClientSocket, 141
WakeVAS::UdpSocket, 137
RealtimeClock
WakeVAS::RealtimeClock, 122
receive
WakeVAS::TcpClientSocket, 140
WakeVAS::UdpSocket, 137
receive_port_
WakeVAS::ClientSystemStateMgr, 86
receive_socket_
WakeVAS::ClientComMgr, 93
WakeVAS::CoreComMgr, 97
received_data_
WakeVAS::TcpClientSocket, 141
WakeVAS::UdpSocket, 138

received_msgs_
  WakeVAS::ClientComMgr, 93
  WakeVAS::CoreComMgr, 97
reciprocal
  WakeVAS::Angle, 163
recv_buffer_
  WakeVAS::TcpClientSocket, 141
  WakeVAS::UdpSocket, 138
ref
  WakeVAS::BDB_RC_Buff, 152
refcount
  WakeVAS::BDB_RC_Buff, 152
refcount_
  WakeVAS::BDB_RC_Buff, 152
RelativeTime
  WakeVAS::RelativeTime, 168, 169
remaining_buffer_size
  WakeVAS::BinaryDataBuffer, 150
removeFault
  WakeVAS::ClientSystemStateMgr, 85
  WakeVAS::SystemStateMgr, 74
report_stream_
  WTMD_CoreApp, 22
reset
  WakeVAS::ClientComMgr, 93
  WakeVAS::ComMgr, 90
  WakeVAS::CoreComMgr, 96
  WakeVAS::CoreRwyStatusMgr, 68
  WakeVAS::CoreSystemStateMgr, 81
  WakeVAS::WfaWrapper, 25
ruc_initializing_
  WakeVAS::WfaWrapper, 26
rwy_buttons
  LocalDialog, 50
  SupervisorDialog, 40
rwy_id_
  WakeVAS::DisableWtmdMsg, 107
  WakeVAS::EnableWtmdMsg, 105
rwy_ids
  LocalDialog, 50
  SupervisorDialog, 40
rwy_mgr_
  WTMD_CoreApp, 23
  WTMD_IDSGwApp, 55
  WTMD_LocalApp, 46
  WTMD_SupervisorApp, 35
rwy_statuses
  LocalDialog, 50
  SupervisorDialog, 40
rwy_statuses_
  WakeVAS::RwyStatusMgr, 64
  WakeVAS::RwyStatusMsg, 103
RwyStatus
  WakeVAS::RwyStatus, 59
RwyStatusList
  WakeVAS::RwyStatusMgr, 63
  WakeVAS::RwyStatusMsg, 102
RwyStatusMgr
  WakeVAS::RwyStatusMgr, 63
RwyStatusMsg
  WakeVAS::RwyStatusMsg, 103
s_half_pi_
  WakeVAS::Angle, 165
s_pi_
  WakeVAS::Angle, 165
s_unref
  WakeVAS::BDB_RC_Buff, 152
SECONDS
  WakeVAS::RelativeTime, 168
seconds_
  WakeVAS::AbsoluteTime, 158
seconds_since_epoch_
  WakeVAS::AbsoluteTime, 158
send
  WakeVAS::ClientComMgr, 92
  WakeVAS::ComMgr, 89
  WakeVAS::CoreComMgr, 96
  WakeVAS::Socket, 132
  WakeVAS::SocketInterface, 128
  WakeVAS::TcpClientSocket, 140
  WakeVAS::TcpServerSocket, 143
  WakeVAS::UdpSocket, 137
send_socket_
  WakeVAS::ClientComMgr, 93
sendHeartbeat
  WakeVAS::ClientSystemStateMgr, 86
  WakeVAS::CoreSystemStateMgr, 81
serial_port_
  WTMD_IDSGwApp, 55
set
  WakeVAS::SimClock, 125
set_sim_time_
  WakeVAS::SimClock, 125
set_wallclock_
  WakeVAS::SimClock, 125
setAsosDirection
  CoreDialog, 31
setAsosSpeed
  CoreDialog, 31
setAvailable
  WakeVAS::CoreRwyStatusMgr, 67, 68
  WakeVAS::RwyStatus, 60
  WakeVAS::RwyStatusMgr, 64
setEnabled
  WakeVAS::ClientRwyStatusMgr, 70
  WakeVAS::CoreRwyStatusMgr, 67, 68
  WakeVAS::RwyStatus, 60
  WakeVAS::RwyStatusMgr, 64
shutdown
  WTMD_CoreApp, 21
  WTMD_IDSGwApp, 54
  WTMD_LocalApp, 45

WTMD_SupervisorApp, 34
SHUTDOWN
WakeVAS::SystemStateMgr, 73
SILENCE_BUTTON_TEXT
SupervisorDialog.cpp, 41, 50
SimClock
WakeVAS::SimClock, 124, 125
sine
WakeVAS::Angle, 163
siteID
WfaWrapper.cpp, 27
Socket
WakeVAS::Socket, 131, 132
Socket.cpp
DEFAULT_BUFFER_SIZE, 135
SocketId
WakeVAS::SocketInterface, 127
SocketInfo
WakeVAS::Socket, 131
WakeVAS::Socket::SocketInfo, 134
SocketInterface
WakeVAS::SocketInterface, 127
sockets_
WakeVAS::SocketInterface, 128
Speed
WakeVAS::Speed, 174, 175
speed_
WakeVAS::AsosWindsMsg, 119
start_time_
WakeVAS::SystemStateMgr::Fault, 77
State
WakeVAS::SystemStateMgr, 73
state_
WakeVAS::CoreSystemStateMgr::ClientData, 83
state_mgr_
WTMD_CoreApp, 23
WTMD_IDSGwApp, 55
WTMD_LocalApp, 46
WTMD_SupervisorApp, 35
status_
WakeVAS::ClientStatusMsg, 112
WakeVAS::CoreStatusMsg, 109
SupervisorAppWindow, 36
~SupervisorAppWindow, 36
closeButtonPressed, 36
SupervisorAppWindow, 36
SupervisorDialog, 37
asos_direction_, 40
asos_speed_, 40
audio_filename_, 40
audio_system_, 40
disableAudioSystem, 40
enableAudioSystem, 40
handleAsosButton, 39
handleAudioButton, 39
handleRwyButton, 39
handleStateDetailButton, 39
handleViewLogButton, 40
rwy_buttons, 40
rwy_ids, 40
rwy_statuses, 40
update, 39
updater_, 41
updateView, 39
SupervisorDialog.cpp
amber, 42, 51
AUDIO_DISABLE_BUTTON_TEXT, 42
AUDIO_DISABLED_STATUS_TEXT, 42
AUDIO_ENABLE_BUTTON_TEXT, 42
AUDIO_ENABLED_STATUS_TEXT, 42
AVAILABLE_AND_ENABLED_STATUS_TEXT, 41, 51
AVAILABLE_STATUS_TEXT, 41
BLANK_BUTTON_TEXT, 41, 50
BLANK_STATUS_TEXT, 42, 51
COORDINATION_PORT_NUM, 41
DISABLE_BUTTON_TEXT, 41
ENABLE_BUTTON_TEXT, 41
ENABLED_ONLY_STATUS_TEXT, 41, 51
FAILED_SYSTEM_STATE_TEXT, 42, 51
INITIALIZING_SYSTEM_STATE_TEXT, 42, 51
OPERATIONAL_SYSTEM_STATE_TEXT, 42, 51
SILENCE_BUTTON_TEXT, 41, 50
TIMER_INTERVAL, 41, 50
UNAVAILABLE_STATUS_TEXT, 41, 51
SupervisorDialog::UpdateTimer, 43
parent_, 43
timerCallback, 43
UpdateTimer, 43
system_state_
WakeVAS::SystemStateMgr, 75
SystemClock
WakeVAS::SystemClock, 121
systemRequestedQuit
WTMD_CoreApp, 21
WTMD_SupervisorApp, 34
SystemStateMgr
WakeVAS::SystemStateMgr, 73
SystemStateMgr.cpp
MAX_LOG_FAULTS, 75
tangent
WakeVAS::Angle, 163
TcpClientSocket
WakeVAS::TcpClientSocket, 140
TcpServerSocket
WakeVAS::TcpServerSocket, 143
terminate_
WakeVAS::TcpClientSocket, 141
WakeVAS::TcpServerSocket, 145

WakeVAS::UdpSocket, 138
time_
WakeVAS::AsosWindsMsg, 119
time_factor_
WakeVAS::SimClock, 125
timer_
WTMD_IDSGwApp, 55
TIMER_INTERVAL
SupervisorDialog.cpp, 41, 50
WTMD_Core.cpp, 23
WTMD_IDSGw.cpp, 56
timerCallback
GwUpdateTimer, 57
LocalDialog::UpdateTimer, 52
SupervisorDialog::UpdateTimer, 43
UpdateTimer, 29
toISO8601
WakeVAS::AbsoluteTime, 157
TX_INTERVAL
CoreRwyStatusMgr.cpp, 68
Type
WakeVAS::Msg, 99
UdpSocket
WakeVAS::UdpSocket, 137
UNAVAILABLE_STATUS_TEXT
SupervisorDialog.cpp, 41, 51
unhandledException
WTMD_CoreApp, 21
WTMD_IDSGwApp, 55
WTMD_LocalApp, 45
WTMD_SupervisorApp, 35
Units
WakeVAS::Angle, 161
WakeVAS::RelativeTime, 168
WakeVAS::Speed, 174
update
LocalDialog, 49
SupervisorDialog, 39
WakeVAS::ClientRwyStatusMgr, 70
WakeVAS::ClientSystemStateMgr, 85
WakeVAS::CoreRwyStatusMgr, 67
WakeVAS::CoreSystemStateMgr, 80
WakeVAS::RwyStatusMgr, 64
WakeVAS::SystemStateMgr, 74
WakeVAS::WfaWrapper, 25, 26
WTMD_CoreApp, 20
WTMD_IDSGwApp, 55
updateClient
WakeVAS::CoreSystemStateMgr, 81
updateConnections
WakeVAS::TcpServerSocket, 144
updater_
LocalDialog, 50
SupervisorDialog, 41
WTMD_CoreApp, 23
UpdateTimer, 29
LocalDialog::UpdateTimer, 52
parent_, 29
SupervisorDialog::UpdateTimer, 43
timerCallback, 29
UpdateTimer, 29
updateView
CoreDialog, 31
LocalDialog, 49
SupervisorDialog, 39
validSocketId
WakeVAS::SocketInterface, 128
value_
WakeVAS::Angle, 165
WakeVAS::RelativeTime, 171
WakeVAS::Speed, 177
WakeVAS::AbsoluteTime, 153
AbsoluteTime, 154, 155
checkDmyhms, 158
computeDmyhms, 158
computeInternalTime, 158
day_of_month_, 159
day_of_year_, 159
equal, 155
getCurrentTime, 158
getDayOfMonth, 157
getDayOfYear, 157
getHour, 157
getJan1_1970_Epoch, 158
getMidnightOfSameDay, 157
getMinutes, 157
getMonth, 157
getSeconds, 157
getTopOfHour, 157
getYear, 156
hour_, 158
jan_1_1970_epoch_, 158
minute_, 158
month_, 159
need_to_compute_dmyhms_, 158
operator-, 156
operator!=, 156
operator+, 156
operator+=, 156
operator<, 156
operator<=, 156
operator-=, 156
operator==, 156
operator>, 156
operator>=, 156
print, 157
printDate, 157
printTime, 157
seconds_, 158
seconds_since_epoch_, 158
toISO8601, 157
year_, 159

WakeVAS::Angle, 160
Angle, 162
cosine, 163
DEGREES, 161
dmsStrToAngle, 164
getDegrees, 162
getDmsStr, 163
getDmStr, 164
getHalfPi, 164
getPi, 164
getPlusMinusDegrees, 163
getPositiveRadians, 162
getRadians, 162
getTensOfDegrees, 163
getWholeDegrees, 163
isValidDmsStr, 164
magnitude, 163
normalize, 165
operator-, 162
operator *=, 162
operator/, 162
operator/=, 162
operator+=, 162
operator-=, 162
POSITIVE_RADIANS, 161
RADIANS, 161
reciprocal, 163
s_half_pi_, 165
s_pi_, 165
sine, 163
tangent, 163
Units, 161
value_, 165
WakeVAS::AsosWindsMsg, 117
~AsosWindsMsg, 118
AsosWindsMsg, 117, 118
direction_, 119
duplicate, 118
getTime, 118
getWindDirection, 118
getWindSpeed, 118
read, 118
speed_, 119
time_, 119
write, 118
WakeVAS::BDB_RC_Buff, 151
~BDB_RC_Buff, 151
BDB_RC_Buff, 151
operator=, 152
p_data, 152
p_storage_, 152
ref, 152
refcount, 152
refcount_, 152
s_unref, 152
WakeVAS::BinaryDataBuffer, 146
~BinaryDataBuffer, 148
add_data, 149
BinaryDataBuffer, 148
clear, 148
current_data_size, 150
data_size, 148
discard_data, 149
extract_data, 148
grow, 149
operator<<, 149
operator=, 148
operator>>, 149
p_data, 148
p_extraction_pt, 150
p_insertion_pt, 150
p_rcbuff_, 150
remaining_buffer_size, 150
write, 149
WakeVAS::ClientComMgr, 91
~ClientComMgr, 92
clearMsgs, 93
ClientComMgr, 92
flushOutput, 92
gatherInput, 92
getReceivedData, 92
initialize, 92
operator=, 93
receive_socket_, 93
received_msgs_, 93
reset, 93
send, 92
send_socket_, 93
WakeVAS::ClientRwyStatusMgr, 69
~ClientRwyStatusMgr, 69
ClientRwyStatusMgr, 69
operator=, 70
setEnabled, 70
update, 70
WakeVAS::ClientStatusMsg, 110
~ClientStatusMsg, 111
ClientStatusMsg, 111
duplicate, 111
failure_, 112
getFailure, 112
getHost, 111
getName, 111
getPort, 112
getStatus, 111
host_, 112
name_, 112
port_, 112
read, 111
status_, 112
write, 111
WakeVAS::ClientSystemStateMgr, 84
~ClientSystemStateMgr, 85

addFault, 85
ClientSystemStateMgr, 85
host_, 86
last_heartbeat_rx_, 86
last_heartbeat_tx_, 86
local_faults_, 86
name_, 86
operator=, 86
putState, 85
receive_port_, 86
removeFault, 85
sendHeartbeat, 86
update, 85

WakeVAS::ComMgr, 88
~ComMgr, 89
ComMgr, 89
flushOutput, 90
gatherInput, 90
getInstance, 89
getReceivedData, 89
instance_, 90
MsgList, 89
operator=, 90
reset, 90
send, 89

WakeVAS::CoreComMgr, 94
~CoreComMgr, 95
addClient, 96
clearMsgs, 96
clients_, 97
ClientSet, 95
CoreComMgr, 95
deleteClient, 96
flushOutput, 96
gatherInput, 96
getInstance, 95
getReceivedData, 95
initialize, 95
instance_, 97
operator=, 96
receive_socket_, 97
received_msgs_, 97
reset, 96
send, 96

WakeVAS::CoreRwyStatusMgr, 66
~CoreRwyStatusMgr, 67
CoreRwyStatusMgr, 67
getInstance, 67
initialize, 67
instance_, 68
last_msg_tx_, 68
need_to_send_, 68
operator=, 68
reset, 68
setAvailable, 67, 68
setEnabled, 67, 68
update, 67

WakeVAS::CoreStatusMsg, 108
~CoreStatusMsg, 109
CoreStatusMsg, 108, 109
duplicate, 109
faults_, 109
getFaultList, 109
getStatus, 109
read, 109
status_, 109
write, 109

WakeVAS::CoreSystemStateMgr, 79
~CoreSystemStateMgr, 80
addLogItem, 81
clients_, 81
ClientSet, 80
CoreSystemStateMgr, 80
last_heartbeat_tx_, 81
operator=, 81
putState, 80
reset, 81
sendHeartbeat, 81
update, 80
updateClient, 81

WakeVAS::CoreSystemStateMgr::ClientData, 83
ClientData, 83
fault_, 83
first_joined_, 83
last_update_, 83
state_, 83

WakeVAS::DisableWtmdMsg, 106
~DisableWtmdMsg, 107
DisableWtmdMsg, 106
duplicate, 107
getRwyId, 107
read, 107
rwy_id_, 107
write, 107

WakeVAS::EnableWtmdMsg, 104
~EnableWtmdMsg, 105
duplicate, 105
EnableWtmdMsg, 104
getRwyId, 105
read, 105
rwy_id_, 105
write, 105

WakeVAS::FaultLogElementMsg, 115
~FaultLogElementMsg, 116
duplicate, 116
fault_, 116
FaultLogElementMsg, 115
getFault, 116
read, 116
write, 116

WakeVAS::FaultLogMsg, 113

~FaultLogMsg, 114
duplicate, 114
fault_log_, 114
FaultLogMsg, 113
getFaultLog, 114
read, 114
write, 114
WakeVAS::Msg, 98
~Msg, 99
duplicate, 100
extract, 100
getType, 100
Msg, 100
msg_type_, 101
read, 100
Type, 99
write, 100
WakeVAS::RealtimeClock, 122
~RealtimeClock, 122
getCurrentTime, 122
operator=, 123
RealtimeClock, 122
WakeVAS::RelativeTime, 167
DAYS, 168
equal, 169
getConversionFactor, 171
getDays, 171
getHours, 170
getInverseConversionFactor, 171
getMilliSeconds, 170
getMinutes, 170
getSeconds, 170
getValue, 170
HOURS, 168
MINUTES, 168
operator-, 170
operator *=, 170
operator!=, 169
operator/, 170
operator/=, 170
operator+, 170
operator+=, 169
operator<, 169
operator<=, 169
operator-=, 170
operator==, 169
operator>, 169
operator>=, 169
RelativeTime, 168, 169
SECONDS, 168
Units, 168
value_, 171
WakeVAS::RwyStatus, 58
~RwyStatus, 59
available_, 60
enabled_, 60
getRwyId, 60
id_, 60
isAvailable, 60
isEnabled, 60
operator!=, 60
operator=, 59
operator==, 59
read, 60
RwyStatus, 59
setAvailable, 60
setEnabled, 60
write, 60
WakeVAS::RwyStatusMgr, 62
~RwyStatusMgr, 63
areAnyAvailable, 64
areAnyEnabled, 63
getInstance, 63
getNumRwys, 63
getRwyIndex, 64
getRwyStatus, 63, 64
instance_, 65
operator=, 64
rwy_statuses_, 64
RwyStatusList, 63
RwyStatusMgr, 63
setAvailable, 64
setEnabled, 64
update, 64
WakeVAS::RwyStatusMsg, 102
~RwyStatusMsg, 103
duplicate, 103
getRwyStatuses, 103
read, 103
rwy_statuses_, 103
RwyStatusList, 102
RwyStatusMsg, 103
write, 103
WakeVAS::SimClock, 124
~SimClock, 124
getCurrentTime, 125
operator=, 125
set, 125
set_sim_time_, 125
set_wallclock_, 125
SimClock, 124, 125
time_factor_, 125
WakeVAS::Socket, 130
~Socket, 131
address_, 133
Direction, 131
getDirection, 132
getReceivedData, 132
info_, 133
initialize, 132
INPUT, 131
INPUT_OUTPUT, 131

makeFullAddress, 132
operator=, 132
OUTPUT, 131
Queue, 131
send, 132
Socket, 131, 132
SocketInfo, 131
WakeVAS::Socket::SocketInfo, 134
~SocketInfo, 134
buffer_size, 135
dir, 134
host_id, 134
port_num, 134
SocketInfo, 134
WakeVAS::SocketInterface, 126
~SocketInterface, 127
addSocket, 128
closeAll, 128
closeSocket, 128
deleteInstance, 127
getNumberOfSockets, 128
getReceivedData, 128
instance_, 129
INVALID_ID, 128
openSocket, 127
openTcpServerSocket, 128
openTcpSocket, 127
operator=, 128
send, 128
SocketId, 127
SocketInterface, 127
sockets_, 128
validSocketId, 128
WakeVAS::Speed, 173
equal, 175
FEET_PER_MIN, 174
FEET_PER_SEC, 174
getConversionFactor, 177
getFPM, 177
getFPS, 176
getInverseConversionFactor, 177
getKnots, 176
getKPH, 176
getMPH, 176
getMPS, 177
getValue, 176
KM_PER_HOUR, 174
KNOTS, 174
magnitude, 176
MPH, 174
operator-, 176
operator *, 176
operator *=, 176
operator!=, 175
operator/, 176
operator/=, 176
operator+=, 175
operator<, 175
operator<=, 175
operator-=, 176
operator==, 175
operator>, 175
operator>=, 175
Speed, 174, 175
Units, 174
value_, 177
WakeVAS::SystemClock, 120
~SystemClock, 121
getCurrentTime, 121
getInstance, 121
instance_, 121
operator=, 121
SystemClock, 121
WakeVAS::SystemStateMgr, 71
~SystemStateMgr, 73
addFault, 74
addLogItem, 74
CORE_TIMEOUT_FAULT, 75
FAILED, 73
fault_log_, 75
FaultList, 72
faults_, 75
getFaultList, 73
getInstance, 73
getState, 73
INITIALIZING, 73
instance_, 75
OPERATIONAL, 73
operator=, 74
putData, 74
putLogData, 74
putState, 74
removeFault, 74
SHUTDOWN, 73
State, 73
system_state_, 75
SystemStateMgr, 73
update, 74
WakeVAS::SystemStateMgr::Fault, 76
end_time_, 77
Fault, 76, 77
getEndTime, 77
getStartTime, 77
noteEndTime, 77
print, 77
read, 77
start_time_, 77
write, 77
WakeVAS::TcpClientSocket, 139
~TcpClientSocket, 140
data_mutex_, 141
endpoint_, 141

getReceivedData, 140
initialize, 140
operator=, 141
read_thread_id_, 141
readThread, 141
receive, 140
received_data_, 141
recv_buffer_, 141
send, 140
TcpClientSocket, 140
terminate_, 141
WakeVAS::TcpServerSocket, 142
~TcpServerSocket, 143
accept, 144
accept_mutex_, 144
accept_thread_id_, 144
acceptThread, 144
ConnectionList, 143
connections_, 144
getReceivedData, 144
initialize, 143
new_peers_, 144
NewPeerList, 143
operator=, 144
peer_acceptor_, 144
send, 143
TcpServerSocket, 143
terminate_, 145
updateConnections, 144
WakeVAS::UdpSocket, 136
~UdpSocket, 137
data_mutex_, 138
endpoint_, 138
getReceivedData, 137
initialize, 137
operator=, 138
read_thread_id_, 138
readThread, 137
receive, 137
received_data_, 138
recv_buffer_, 138
send, 137
terminate_, 138
UdpSocket, 137
WakeVAS::WfaWrapper, 24
~WfaWrapper, 25
getInstance, 25
init_time_, 26
initialize, 25
instance_, 26
operator=, 26
reset, 25
ruc_initializing_, 26
update, 25, 26
WfaWrapper, 25
was_available_
WTMD_CoreApp, 22
was_operational_
WTMD_CoreApp, 22
wfa_
WTMD_CoreApp, 23
WfaWrapper
WakeVAS::WfaWrapper, 25
WfaWrapper.cpp
alertDataSet, 27
alertParams, 27
currProf, 27
currWind, 27
MAX_ASOS_AGE, 27
MAX_ASOS_AGE_SECS, 26
MAX_RUC_AGE, 27
siteID, 27
windPredData, 27
windPredParams, 27
window_
WTMD_CoreApp, 23
WTMD_LocalApp, 46
WTMD_SupervisorApp, 35
windPredData
WfaWrapper.cpp, 27
windPredParams
WfaWrapper.cpp, 27
write
WakeVAS::AsosWindsMsg, 118
WakeVAS::BinaryDataBuffer, 149
WakeVAS::ClientStatusMsg, 111
WakeVAS::CoreStatusMsg, 109
WakeVAS::DisableWtmdMsg, 107
WakeVAS::EnableWtmdMsg, 105
WakeVAS::FaultLogElementMsg, 116
WakeVAS::FaultLogMsg, 114
WakeVAS::Msg, 100
WakeVAS::RwyStatus, 60
WakeVAS::RwyStatusMsg, 103
WakeVAS::SystemStateMgr::Fault, 77
WTMD_Core.cpp
TIMER_INTERVAL, 23
WTMD_CoreApp, 19
~WTMD_CoreApp, 20
alert_ostream_, 22
asos_istream_, 22
available_duration_total_, 22
available_time_, 22
clock_, 22
closeoutReport, 21
com_mgr_, 22
getApplicationName, 21
getApplicationVersion, 21
handle, 20
initialise, 21
instance, 20
instance_, 23

moreThanOneInstanceAllowed, 21
offline_, 21
operational_duration_total_, 22
operational_time_, 22
operator=, 21
profile_istream_, 22
report_stream_, 22
rwy_mgr_, 23
shutdown, 21
state_mgr_, 23
systemRequestedQuit, 21
unhandledException, 21
update, 20
updater_, 23
was_available_, 22
was_operational_, 22
wfa_, 23
window_, 23
WTMD_CoreApp, 20
WTMD_IDSGw.cpp
TIMER_INTERVAL, 56
WTMD_IDSGwApp, 53
~WTMD_IDSGwApp, 54
clock_, 55
com_mgr_, 55
getApplicationName, 54
getApplicationVersion, 54
initialise, 54
moreThanOneInstanceAllowed, 54
operator=, 55
rwy_mgr_, 55
serial_port_, 55
shutdown, 54
state_mgr_, 55
timer_, 55
unhandledException, 55
update, 55
WTMD_IDSGwApp, 54
WTMD_LocalApp, 44
~WTMD_LocalApp, 45
clock_, 46
com_mgr_, 46
getApplicationName, 45
getApplicationVersion, 45
initialise, 45
moreThanOneInstanceAllowed, 45
name_, 46
operator=, 46
rwy_mgr_, 46
shutdown, 45
state_mgr_, 46
unhandledException, 45
window_, 46
WTMD_LocalApp, 45
WTMD_SupervisorApp, 33
~WTMD_SupervisorApp, 34
clock_, 35
com_mgr_, 35
getApplicationName, 34
getApplicationVersion, 34
initialise, 34
moreThanOneInstanceAllowed, 34
name_, 35
operator=, 35
rwy_mgr_, 35
shutdown, 34
state_mgr_, 35
systemRequestedQuit, 34
unhandledException, 35
window_, 35
WTMD_SupervisorApp, 34
year_
WakeVAS::AbsoluteTime, 159

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 01- 12 - 2008	**2. REPORT TYPE** Contractor Report	**3. DATES COVERED** *(From - To)*

4. TITLE AND SUBTITLE Wake Turbulence Mitigation for Departures (WTMD) Prototype System - Software Design Document	**5a. CONTRACT NUMBER** L70750D
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Sturdy, James L.	**5d. PROJECT NUMBER**
	5e. TASK NUMBER Task Assignment 139
	5f. WORK UNIT NUMBER 305295.02.07.07.20
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center, Hampton, VA 23681-2199; Raytheon Company, 130 Research Drive, Hampton, VA 23666	**8. PERFORMING ORGANIZATION REPORT NUMBER** CONITS TP 139
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	**10. SPONSOR/MONITOR'S ACRONYM(S)** NASA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) NASA/CR-2008-215549

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 03
Availability: NASA CASI (301) 621-0390

13. SUPPLEMENTARY NOTES
Langley Technical Monitor: Cornelius J. O'Connor

14. ABSTRACT
This document describes the software design of a prototype Wake Turbulence Mitigation for Departures (WTMD) system that was evaluated in shadow mode operation at the Saint Louis (KSTL) and Houston (KIAH) airports. This document describes the software that provides the system framework, communications, user displays, and hosts the Wind Forecasting Algorithm (WFA) software developed by the M.I.T. Lincoln Laboratory (MIT-LL). The WFA algorithms and software are described in a separate document produced by MIT-LL.

15. SUBJECT TERMS
Aircraft wakes; Computer programs

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	200	(301) 621-0390

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18